VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Tomáš Staněk

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Staněk**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: MonkeyData, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Ing. Rostislav Kreisinger

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 15. dubna 2019

MONKEYDATA
MonkeyData s.r.o. -4-
Hladnovská 1255/23, 710 00 Ostrava
IČ: 02731452 DIČ: CZ02731452
www.monkeydata.cz

ING. JAN LASTŮVKA

Děkuji firmě MonkeyData s.r.o. a Ing. Rostislavu Kreisingerovi za poskytnutí možnosti absolvování individuální odborné praxe. Rád bych poděkoval těm, kteří mi konzultací a radou pomohli, především všem svým kolegům a také manželce za její trpělivost při vzniku této práce. V neposlední řadě bych rád poděkoval doc. Mgr. Jiřímu Dvorskému, Ph.D. za jeho lidský přístup, konzultace, vedení a odbornou pomoc při zpracování této práce.

Abstrakt

V této bakalářské práci je popsána autorova odborná praxe ve společnosti MonkeyData s.r.o., její průběh, uplatnění znalostí a dovedností získaných během autorova studia. V úvodu je představena společnost MonkeyData s.r.o. včetně vzniku a krátké historie. Dále jsou popsány důvody volby firmy a také představení aplikace a její potřeby. Poté jsou rozebrány řešené úkoly, na kterých autor během vývoje pracoval včetně použitých technologií. V závěru práce je zhodnocení použitých a chybějících dovednosti, nově nabytých znalostí i celkový průběh praxe.

Klíčová slova: bakalářská práce, vývoj, webová aplikace, MonkeyData, PHP, MySQL, XtraDB, OAuth 2.0

Abstract

In this bachelor thesis is described author's practice in MonkeyData s.r.o., its course, application of knowledge and skills acquired during the author's study. The introduction introduces MonkeyData s.r.o. including the origin and short history. There are also described the reasons for choosing a company as well as the application and its needs. Then there are solved tasks, on which the author worked during the development including used technologies. At the end of the work is the evaluation of used and missing skills, newly acquired knowledge and general practice.

Key Words: bachelor thesis, development, web application, MonkeyData, PHP, MySQL, XtraDB, OAuth 2.0

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CAPTCHA	–	Completely Automated Public Turing test to tell Computers and Humans Apart[6]
CE	–	Community Edition
CI	–	Continues Integration
CRUD	–	Create, Read, Update, Delete
DB	–	Database
GTID	–	Global Transaction Identifier
HTTP(S)	–	Hypertext Transfer Protocol (Secured)
IETF	–	Internet Engineering Task Force
IP	–	Internet Protocol
IPC	–	Inter-Process Communication
IS	–	Information System
IT	–	Information Technology
JSON	–	JavaScript Object Notation
LB	–	Load Balancer
LTS	–	Long Term Support
L4	–	Layer 4
MB	–	Megabyte
OOM	–	Out Of Memory
OOP	–	Object Oriented Programming
PHP	–	PHP: Hypertext Preprocessor[19]
REST	–	Representational State Transfer
RFC	–	Request For Comments
SIGINT	–	IPC Signal Interruption
SIGKILL	–	IPC Signal Kill
SQL	–	Structured Query Language
SSO	–	Single sign-on
TCP	–	Transmission Control Protocol
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
USA	–	The United States of America
VM	–	Virtual Machine
VŠB-TU	–	Vysoká Škola Báňská - Technická Univerzita
XML	–	eXtensible Markup Language

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce vznikla jako završení odborné praxe vykonané ve firmě MonkeyData s.r.o. během mého studia na VŠB-TU Ostrava. Pro možnost vykonání odborné praxe ve firmě jsem se rozhodl na základě doporučení a konzultace paní tajemnice a myslím si, že to byla nejlepší volba. Již před praxí jsem se seznámil s prostředím firmy a přišlo mi velmi zajímavé aplikovat znalosti nabyté studiem na VŠB-TU Ostrava v rychle se vyvíjejícím prostředí firmy označované jako mezinárodní technologický startup. V první části práce se seznámíme s krátkou historií a prostředím firmy MonkeyData. Dále popisuji aplikaci, na které jsem pracoval, její potřebu a jednotlivé úkoly, které jsem řešil. Ke konci práce shrnuji znalosti a nabyté studijní zkušenosti, které jsem během odborné praxe využil, a také znalosti, které při studiu chyběly anebo byly zmíněny jen okrajově. V závěru práce hodnotím odbornou praxi a její průběh.

2 O společnosti MonkeyData

Obrázek 1: Logo MonkeyData s.r.o.

Firma MonkeyData vznikla na začátku roku 2014, ale prvotní nápad a vlastně i samotné úvahy o vytvoření analytické aplikace pro eshopy se začaly formovat již na konci předešlého roku. Na počátku zrodu společnosti MonkeyData stáli tři zakladatelé, bývalí spolužáci z vysoké školy, kteří si po studiu ekonomie založili vlastní eshop. Díky této praktické zkušenosti brzy zjistili, že na trhu neexistuje produkt, který by umožnil majitelům eshopů spravovat marketingová a salesová data z různých nástrojů na jednom místě. Původně se tedy firma zabývala hlavně vývojem a provozem stejnojmenné aplikace, která měla jako hlavní cíl analýzu a zpracování dat z internetových obchodů a dalších marketingových kanálů a následné zobrazení zpracovaných dat uživatelům (zpravidla majitelé a administrátoři eshopů, ne koncoví zákazníci obchodů) v jednoduché a srozumitelné podobě v přehledných grafech. Vzhledem k vývoji trhu a nutnosti držet krok s eCommerce trendy postupně přibyly k webové aplikaci také aplikace mobilní. Ačkoli se postupným vývojem produkty částečně měnily, směr zůstal stále stejný. I nyní se společnost zabývá analýzou, zpracováním a následnou prezentací eshopových dat. Aktuálně se jedná převážně o formu analytických pluginů a aplikací integrovaných přímo do administrace eshopových platforem. Společnost má sídlo v Ostravě a v Austinu (Texas, USA) a se svým produktovým portfoliem expandovala do zahraničí, kde se zaměřuje především na evropský trh a USA. K dnešnímu dni zaměstnává zhruba 20 odborníků z oblasti IT, designu, marketingu a spolupracuje také s externími zaměstnanci a studenty. Od svého vzniku se společnost MonkeyData zúčastnila mnoha mezinárodních konferencí a akcí zaměřených právě na oblast online obchodování a analýzy dat (viz příloha A). V rámci spolupráce s eshopovou platformou Shopify pořádali meetup v Praze a zapojili se také do programu podpory inovací a růstu startupů v Ostravě, kde pořádali ve spolupráci s Centrum PANT několik meetupů a workshopů.

3 Důvody pro volbu firmy

Při volbě firmy jsem hlavně zohledňoval její zaměření a částečně i velikost. Firem vyvíjejících eshop XY je spousta, ale já neměl zájem o repetitivní činnosti spojené s vývojem eshopových řešení. Pracovat v takové firmě se možná zdá býti výhodné minimálně co se stability na trhu týče, ale většinou to je spíše kolos se zaběhnutým systémem, kde není velký prostor pro inovace a změnu. A v tomto směru mne firma MonkeyData oslovila naplno. Inovace, a s tím související vývoj a testy nových systémů a technologií, je přímo firemní hodnota číslo jedna. Podle hesla „Fail fast, learn faster“ je zde prostor pro chyby a hledání toho lepšího řešení. A já chci hledat lepší řešení a spolupráci s MonkeyData jsem cítil jako šanci tento cíl si plnit. Firma také hodně využívá open source projektů a aplikací a různým způsobem se podílí na jejich vývoji. I zde jsem našel shodu. Sám do open source a komunitních projektů přispívám a velmi rád se v tomto směru budu angažovat i v kolektivu.

3.1 Pracovní pozice

Před nástupem do firmy mě kontaktoval jeden z majitelů jako reakce na můj email a pozval na pracovní pohovor. Tématem setkání bylo zjistit nejen mé zkušenosti z oboru IT, ale i osobnostní stránku. Po tomto evidentně úspěšném prvním kole jsem dostal krátký test sestávající z několika různorodých problémů a já měl najít řešení. I zde vše proběhlo bez potíží a já byl přijat na pozici programátora.

3.2 Prostředí a nástroje

K interní komunikaci se převážně používá Slack[24], který umožňuje posílání zpráv napřímo, udržování komunikace ve vláknech a také v tématických kanálech, je možné vyhledávat v historii a obsahuje spoustů botů a rozšíření pro napojení na další služby pro zjednodušení práce, jak jednotlivce, tak i skupin. Asi nejpoužívanějším doplněkem je z mého pohledu Slack Webhook, kterým je možno zaslat zprávu v podstatě odkudkoliv. Byť i Slack umožnuje volání, tak k firemním hovorům se převážně používá Google[10] Meet a to díky jeho integraci do Google Kalendáře, který je nedílnou součástí života všech ve firmě, a to jak interních lidí, tak externistů. V mnoha případech jsou výše uvedené nástroje pro komunikaci pomocí, avšak nic nenahradí osobní kontakt, který je na denní bázi a využívá se prostřednictvím schůzek a mítinků k debatám, získávání informací a řešení problémů. Nutno podotknout, že se v mnoha případech ukázalo, že osobní schůzka a obrázek na prezentaci je mnohem více než tisíc slov ve Slacku.

Co se vývojových nástrojů týče tak základem všeho je Git[9], kde je možné využívat několika větví zároveň, například pro paralelní vývoj, a také mít speciální větve, třeba zvlášť pro produkci anebo nestabilní testy. Firma využívá svůj vlastní git server, konkrétně Gitlab CE, který je webovou nádstavbou nad gitem a umožňuje pohodlnou správu repozitářů jakožto i správu uživatel a různých rozšíření a dodatků pro moderní vývoj. Nedílnou součástí vývoje jsou i nástroje

Jenkins[12] a Ansible[2], a to zejména proto, že spolu s gitem tvoří firemní CI, jenž usnadňuje práci programátorům obzvlášť při vývoji a testech. Z dalších použitých nástrojů zmíním Jira a Confluence, které oba pocházejí z dílny firmy Atlassian[4] a slouží pro správu úkolů a jejich flow, respektive pro správu projektů. Tyto nástroje umožňují vzájemnou integraci, což je velká pomoc při řešení více projektů zároveň především díky možnému odkazování na úkoly, poznámky a přílohy.

4 O aplikaci

Tématem mé odborné praxe byl vývoj interní webové aplikace, která by měla sloužit jako systém pro správu uživatel a klientů, jejich autentizaci a autorizaci při přístupu ke zdrojům a službám. Byl jsem zařazen do týmu, jenž měl na starosti kromě tohoto projektu ještě několik dalších. Celý tým se skládal z osmi lidí, přičemž dva se zabývali vývojem mobilních aplikací, další dva řešili importní systémy a zbylí čtyři, včetně mne, se starali o samostatné projekty, které byly vyvíjeny jako mikroslužby do ekosystému aplikací provozované firmou MonkeyData. Většina aplikací v MonkeyData není vyvíjena pro externí zákazníky, ale jsou zadávány vedením firmy anebo jsou vyvíjeny na základě požadavků diktovaných interními potřebami. Můj projekt vznikl jako interní potřeba. Existoval již neformální soupis požadavků toho, co by měla aplikace umět a kdo a jak by měl systém používat, a proto před samotným vývojem byla uskutečněna jen jedna delší schůzka, kde bylo přítomno vedení firmy, kolegové z vývoje a také zástupci grafického oddělení a marketingu. Na tomto mítinku byla vytvořena vize služby a vyplynula spousta informací, které bylo nutno zpracovat a na základě kterých se posléze postupovalo při návrhu a vývoji. Bylo také určeno kódové označení tohoto systému: „AUTH server," které mělo sloužit jako kratší a jedinečný název používaný při komunikaci s ostatními členy týmu.

4.1 Potřeba systému

Postupným vývojem se z jedné aplikace MonkeyData s vlastní správou uživatel stal malý ekosystém aplikací, kde řízení přístupu, přihlašování a identifikace uživatele byla nejednotná. Každá aplikace a část s vlastní databází uživatel byla sama sobě autoritou, nebylo možné přesně identifikovat stejného uživatele jiných aplikací a i uživatelé, kteří používali více aplikací rodiny MonkeyData, byli nuceni mít několik uživatelských profilů. Se zaměřením firmy na analytické pluginy a ve větší míře i na mobilní aplikace došlo k nárůstu uživatelské báze a také k nevyhnutelnému, bylo nutné sjednotit řízení a správu uživatel, ideálně vše zjednodušit a vytvořit základ pro systém služeb, které budou spadat pod jednu autoritu a ověřovat své požadavky, ale i komunikaci mezi sebou. To vše spolu se zabezpečenou komunikací obecně a důrazem na ochranu osobních údajů zákazníků a jejich klientů a obchodních partnerů.

5 Autentizační a autorizační systém

Autentizační a autorizační systém by měl být centrálním zabezpečovacím prvkem. Hlavním cílem tohoto projektu je vytvoření vysoce dostupné služby stojící jako autorita pro ekosystém dalších aplikací. Tato platforma bude využita hlavně pro registraci a přihlašování uživatel, autentizaci a autorizaci jak uživatel, tak i klientů třetích stran. Vzhledem k internímu zaměření této služby budou jako klienti třetích stran převážně další firemní systémy, pro které by mělo být dostupné i použití SSO. Celá aplikace se dá rozdělit na pět částí, jenž svou funkcionalitou řeší různé oblasti a způsoby použití. Jedna část obstarává webové API, další řeší přístup pomocí rozšířeného protokolu OAuth 2.0 a zbylé tři části se týkají odlišných uživatelských rolí.

5.1 Část 1 - Webové API

Tato část systému slouží především k přenosu informací, je orientována datově a využívá REST postavený nad protokolem HTTPS. Jedná se o rozhraní, kterým mohou komunikovat dvě aplikace mezi sebou. V našem případě jde hlavně o jednosměrnou komunikaci směrem ven, což umožňuje dalším systémům především ověření informací, se kterými pracují v rámci ekosystému aplikací. Pro úspěšnou komunikaci je nutnou podmínkou mít platný API token, který lze získat několika způsoby. Každý API token je v systému unikátní, je časově omezený a je vázán na registrovaného uživatele a také na aplikaci třetí strany spolu s příslušnými právy. Pro standardní průběh získání API tokenu je pro aplikaci třetí strany nutný i souhlas vlastníka zdrojů, což je v našem případě již zmíněný registrovaný uživatel.

5.2 Část 2 - OAuth2

Část OAuth2 implementuje OAuth 2.0 Authorization Framework podle specifikace RFC6749[22] a Bearer Token Usage RFC6750[23] definované IETF. Tato část systému umožňuje jiným systémům (klientům, aplikacím třetích stran) získání limitovaného přístupu ke zdrojům vlastníka bez toho, aniž by vlastník musel jakýmkoliv způsobem kompromitovat své přihlašovací údaje. Zde má klient na výběr z několika možností (grant), jak získat přístupový token. Každý klient musí být registrovaný a při přístupu se musí prokázat platnými údaji. Klientům s neplatnými klientskými údaji není přístup povolen a nemohou získat přístupový token. Klient se nemůže registrovat automaticky, tuto registraci je nutné provést jako přihlášený uživatel, který může založit klientský účet, kde mu posléze budou zobrazeny klientské přístupové údaje.

Z důvodu kompatibility starších systémů byly nad rámec základní specifikace RFC6749 přidány i další metody, které umožňují interní aplikaci zprostředkování přihlášení a registrace. I zde ale platí, že klient musí být registrován, aby mohl toto zprostředkování provést. Při tomto režimu platí stejný postup a pravidla jako u neregistrovaného uživatele s výjimkou ověření emailové adresy uživatele při registraci, uživatelův email byl již ověřen, a proto se neposílá registrační email s odkazem obsahujícím náhodně vygenerovaný token pro ověření pravosti emailového účtu.

5.3 Část 3 - Neregistrovaný uživatel

Do této části aplikace se dostane každý nepřihlášený anebo neregistrovaný uživatel. Těmto uživatelům není přístupné nic jiného než přihlašovací stránka, kde se mohou do aplikace přihlásit, a registrační stránka, kde se mohou do aplikace registrovat. Pro registraci je povinným údajem uživatelův email, který slouží nejen jako přihlašovací jméno, ale také pro komunikaci s uživatelem. Při registraci může být potřeba prokázat, zda-li je přítomen člověk a úspěšně projít přes CAPTCHA. Po úspěšné registraci je na tuto emailovou adresu odeslán registrační email s odkazem obsahujícím náhodně vygenerovaný token pro ověření pravosti emailového konta. Bez ověření emailové adresy se nelze do aplikace přihlásit. Při registraci uživatel zadává i zvolené přístupové heslo, které musí splnit základní požadavky složitosti a délky. Uživatelovo heslo není uloženo, nelze je tedy získat. V případě zapomenutí uživatelského hesla je možné využít obnovy hesla a získat jednorázový časově omezený přístup prostřednictvím odkazu poslaného na uživatelův email vedený u jeho profilu v aplikaci a nastavit si nové heslo. Odkaz pro obnovu hesla je na přihlašovací stránce.

5.4 Část 4 - Registrovaný uživatel

Uživatel se po registraci a ověření emailové adresy prostřednictvím registračního emailu s odkazem může do aplikace přihlásit. Po přihlášení má uživatel k dispozici všechny dostupné funkce, po ověření uživatelova emailu není nutné žádné dodatečné schválení uživatelského účtu administrátorem. Přihlášený uživatel má k dispozici menu, díky kterému má možnost přesunu do jednotlivých kategorií pro správu účtu a pohodlné ovládání. Menu, které je zobrazeno jako lišta v horní části stránky, je rozděleno na několik částí. Navigace je zobrazena vlevo nahoře a zde má uživatel přístupné ovládání a správu klientů a založení nového klienta pomocí jednoduchého formuláře, kde lze vložit i obrázek či logo, kterým bude jeho klient reprezentován. Dále v této sekci je možné spravovat všechny vlastní klienty, upravit popisek, doplnit či změnit jejich logo, vidět počty připojených uživatel k jednotlivým klientům a také vidět seznam cizích klientů, kterým uživatel dal nějaký druh přístupu. Uživatel má možnost si zobrazit u jednotlivých cizích klientů jejich rozsah přístupu a může i zrušit přístup těmto klientům v případě potřeby. Další položka navigace vlevo je aktivita a zobrazení přístupů klientů v chronologickém pořadí s možností filtrování podle rozličných parametrů. Každý uživatel má v horní pravé části navigace ikonu s jeho profilovou fotkou, která otevírá dropdown menu. V této sekci je nastavení uživatelského profilu, kde lze upravit veškeré údaje o svém profilu, například přidat či změnit profilový obrázek anebo změnit heslo. Další položkou v dropdown menu je nastavení. V nastavení si uživatel může upravit vzhled aplikace, národní preference a formát zobrazení čísel a data. Poslední položkou tohoto menu je odhlášení.

5.5 Část 5 - Administrátor

Administrátorský účet je nutné do aplikace nastavit za použití příkazu v systémové konzoli, kvůli bezpečnosti jej nelze pomocí webového rozhraní jakkoliv přidat či upravit. Administrátor má možnost spravovat všechny registrované uživatele a klienty. Hlavní náplní administrátora je dohlížení nad celou aplikací a vyřizování případných požadavků uživatel, například smazání profilu anebo jeho obnovení, pokud je to možné.

Horní menu se v případě administrátorského účtu liší jen minimálně, administrátor má k dispozici vše, co i standardní uživatel. Přibyla pouze ikona ozubeného kolečka, která otevírá levé boční menu s položkami pro správu celé aplikace. Hlavními položkami správy jsou celkový přehled, monitoring a nastavení. V přehledu má administrátor k dispozici sumář stavu uživatel a klientů a sekce pro správu všech klientů a sekci pro správu všech uživatel. V klientské sekci má administrátor náhled na klienty a možnost je filtrovat a zobrazit podle různých kritérií. U každého záznamu klienta lze prohlížet jeho profil, vlastníka klienta, připojené uživatele a take celkový seznam povolených přístupů. Administrátor může zakázat klienta na globální úrovni v případě z podezření porušování pravidel. V ten moment již klient nemůže přistupovat ke zdrojům uživatel a je nutné obnovit registraci daného klienta, což již není možné udělat automaticky, ale pouze skrz administrátora. V uživatelské sekci je přístupný seznam všech uživatel a náhled na jejich profil, spolu s jejich vytvořenými klienty, kde se administrátor může proklikem dostat do klientské sekce na profil konkrétního klienta. V monitoringu může administrátor sledovat parametry aplikace jako jsou počty uložených záznamů apod. Sekce nastavení zatím nemá žádné použití, ale počítá se s budoucím rozšířením pro nastavení visuálu a dalších prvků ovlivňujících chování aplikace.

6 Řešené úkoly

6.1 Časová náročnost

Úkol	Odhad	Skutečnost
1. Analýza a návrh systému	15 dnů	15 dnů
2. Implementace aplikace	15 dnů	17 dnů
3. Výběr úložiště	8 dnů	10 dnů
4. Mobilní aplikace	5 dnů	5 dnů
5. Kontejnerizace aplikace	5 dny	4 dny

6.2 Úkol č.1 - Analýza a návrh systému

6.2.1 Zadání úkolu

Mým prvním úkolem bylo provést analýzu a celkový návrh systému na základě informací z firemního mítinku, který se konal jako pilotní schůzka pro získání potřebných podkladů pro vývoj. Cílem tohoto návrhu bylo především vytvoření specifikace pro následnou implementaci systému se zohledněním role, do které bude tento systém postaven. Důraz byl kladen především na vytvoření specifikace zahrnující hlavně funkčnost týkající se části aplikace OAuth2, která je považována za základní jádro aplikace.

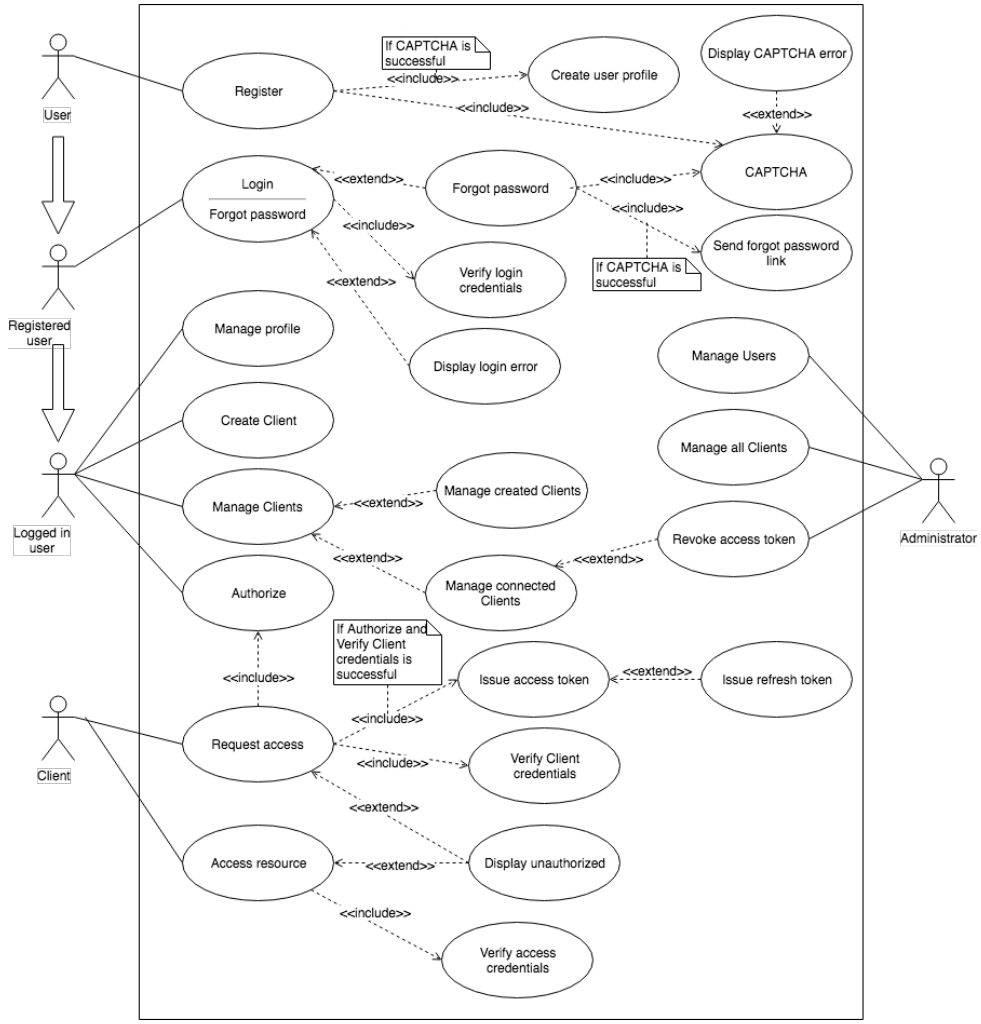
6.2.2 Rozbor problému

Před samotným návrhem bylo nutné si znova projít veškeré materiály a poznámky z firemního mítinku a utříbit a definovat základní požadavky navrhovaného informačního systému. I přes jistou znalost protokolu OAuth2 bylo potřeba si opět pročíst celou specifikaci RFC6749 a co nejlépe se seznámit se všemi body. Zde jsem se zaměřil především na samotný protokol, jednotlivé endpointy potřebné pro funkci a informace o jednotlivých typech autorizace a také na zde definované role. Nastudoval jsem si i specifikaci RFC6750, která se zabývá použitím přístupového tokenu.

6.2.3 Realizace

Při navrhování aplikace jsem vycházel z metodik a postupů vývoje informačních systémů a začal tedy postupně analyzovat jednotlivé části aplikace. Po teoretické přípravě jsem přešel k samotné realizaci návrhu a to definicí jednotlivých aktérů, kteří vstupují do procesů, respektive je provádějí. Vodítkem pro mne bylo i rozdělení aplikace do několika částí, z toho mi vyplynulo základní dělení uživatelských rolí a také rozdíly mezi jednotlivými rolemi. Jako první aktér je zde neregistrovaný uživatel, který má k dispozici pouze možnost se zaregistrovat. Po úspěšné registraci

máme registrovaného uživatele s možností přihlašení a konečně registrovaného přihlášeného uživatele vystupujícího i v roli vlastníka klientů. Speciálním případem registrovaného uživatele je administrátor. Vytvořil jsem UseCase Diagram zobrazující tyto role včetně klienta, jehož dva hlavní případy použití jsou požadavek přístupu a přístup ke zdrojům uživatele.



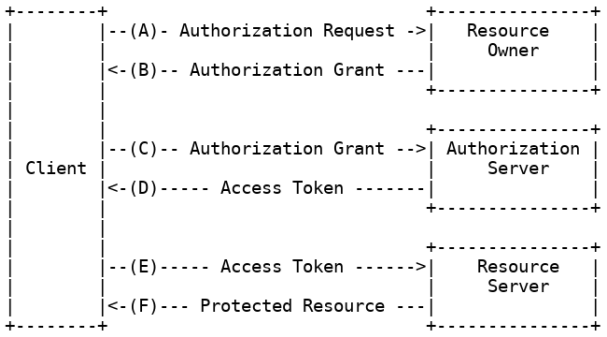
Obrázek 2: Use Case diagram

Na obrázku 2 vidíme tyto aktéry:

- User je anonymní uživatel, který se může zaregistrovat.
- Registered user je již registrovaný uživatel, který se může přihlásit.
- Logged in user je přihlášený uživatel vystupující v roli uživatele, vlastníka zdrojů a také vlastníka klientů.
- Client je registrovaná aplikace třetí strany.
- Administrátor.

Při definici aktérů jsem vycházel ze specifikace OAuth 2.0, kde jsou zmíněny čtyři druhy rolí. Jsou to vlastník zdrojů, klient, server poskytující zdroje vlastníka a autorizační server. S konzultantem a dalšími členy týmu proběhlo několik menších sezení, která měla za cíl validovat mnou navržené role a aktéry, kde bylo nutné zohlednit a zakomponovat pozice a napojení dalších firemních systémů. Ty jsem rozdělil na služby, které běží na pozadí a uživatel, jak interní tak externí, se k nim nemůže přihlásit, a ty, které mají webové rozhraní a jsou de facto určeny jako frontend pro ovládání a správu služeb běžících na pozadí anebo jako koncové aplikace pro potřeby externích uživatel. Pro dodržení specifikace OAuth 2.0 bylo potřeba se příliš nevzdálit od konceptu, a proto i mnou definovaní aktéři korespondují s jednotlivými rolemi ze specifikace OAuth 2.0. Vlastník zdrojů je náš registrovaný uživatel, který je schopný přidělit nebo odepřít přístup ke chráněnému zdroji. Klient OAuth 2.0 je aplikace třetí strany, v našem případě to jsou převážně jiné interní systémy a aplikace reprezentované profilem klienta registrovaného v našem systému, a server poskytující zdroje vlastníka je uvažován jako aplikace nebo systém poskytující službu či data uživatele. A nakonec role autorizační server z OAuth 2.0 je zastřešena části OAuth2 navrhovaného systému.

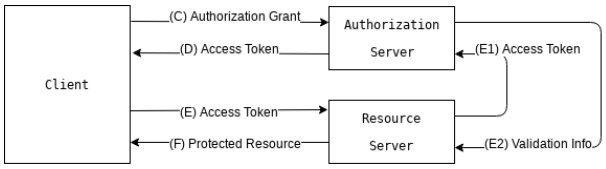
Po formulaci aktérů následovala analýza jednotlivých případů použití se zaměřením na požadavek přístupu a přístup ke zdrojům vlastníka. Abstraktní flow požadavku přístupu vidíme na obrázku 3. Kroky (C) a (D) znázorňují komunikaci klienta s autorizačním serverem včetně předchozího schválení požadavku vlastníkem zdrojů, kroky (A) a (B).



Obrázek 3: OAuth 2.0 Abstract Protocol Flow[22]

V navrhovaném systému se předpokládá, že přístup klienta ke zdrojům vlastníka je volání endpointu webového API aplikace poskytující zdroje vlastníka s přístupovým tokenem. Obecné flow lze vidět na obrázku 3 znázorněné kroky (E) a (F). Standard OAuth 2.0 umožňuje, aby autorizační server a server poskytující zdroje vlastníka byly dvě oddělené služby, což kromě případu zdrojů poskytovaných webovým API autorizačního serveru, například profil uživatele, je každý další případ. Standard ale nespecifikuje, jakým způsobem a jak budou v takovém případě dvě oddělené aplikace komunikovat. Při návrhu jsem zohlednil i tento případ, který bude velmi častý v interním ekosystému aplikací, jelikož každá aplikace by měla běžet jako samostatná služba, která nebude mít možnost přistupovat do úložiště autorizačního serveru a nebude moci přímo ověřit platnost přístupového tokenu. Toto chování by bylo nechtěné a značně by narušilo

zapouzdření systému a kompetence autorizačního serveru. Vzniklo proto rozšířené flow, obrázek 4, kde je znázorněna komunikace mezi aplikací poskytující zdroje vlastníka a autorizačním serverem, kroky (E1) a (E2).



Obrázek 4: Extended Abstract Protocol Flow

Aplikace poskytující zdroje vlastníka musí při požadavku na přístup ke zdrojům ověřit platnost přístupového tokenu a také validitu a platnost rozsahu přístupu, krok (E1). Na webovém API autorizačního serveru proto bude speciální endpoint, který bude pro konkrétní přístupový token vracet informace o tomto tokenu, krok (E2), mimo jiné platnost a expiraci včetně identifikace klienta, pro kterého byl tento token vydán, a také rozsah povolení vázaných k tomuto tokenu. Pokud nebude token z nějakého důvodu platný, autorizační server vrátí chybovou odpověď.

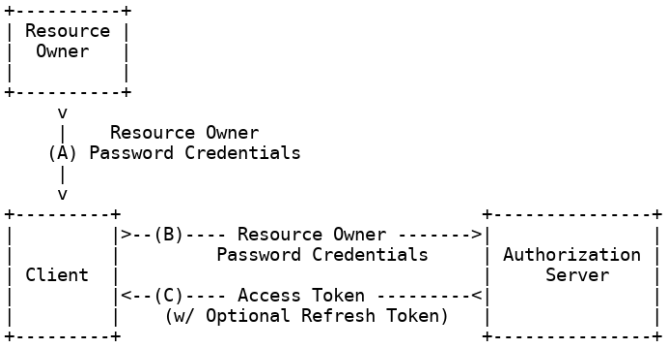
Ujasnění si a pochopení fungování komunikace standardu OAuth 2.0 bylo klíčové pro správný návrh objektů a komponent aplikace. Kromě navrhnutých aktérů a rolí bylo stěžejní projít i jednotlivé možnosti získání přístupu (Grant) definované v OAuth 2.0 a jejich potřebnost ve vznikajícím systému. Dle specifikace není nutné implementovat všechny typy získání přístupu, autorizační server může podporovat jen některé z nich.

Standard OAuth 2.0 definuje čtyři základní typy metod získání přístupového tokenu.

- Authorization Code Grant
- Implicit Grant
- Resource Owner Password Credentials Grant
- Client Credentials Grant

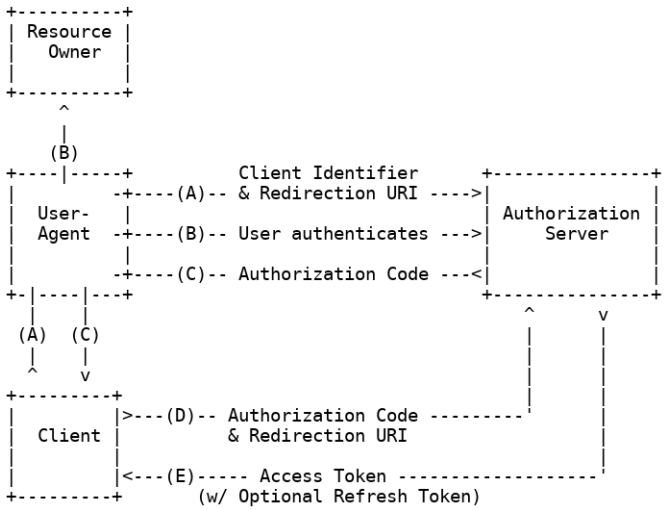
Vzhledem k primárnímu použití AUTH serveru jako autority pro ekosystém interních aplikací provozovaných na zabezpečených serverech jsem úplně vynechal podporu pro Implicit Grant, který je sice vhodný pro použití ve webových prohlížečích, ale je nutné, aby uživatel při požadavku na vydání tokenu vždy zadával přihlašovací údaje, protože není bezpečné uchovávat nebo jakkoliv distribuovat secret klienta do veřejné části webu. Byla by tím narušena identita klienta a zabezpečení zdrojů uživatel. S údaji klienta, klientské id a secret, by útočník mohl získat tímto způsobem přístupový token pro jiného uživatele.

Resource Owner Password Credentials Grant také vyžaduje zadání přihlašovacích údajů uživatele, obrázek 5 krok (A), a je vhodný zvlášť pro použití v interních aplikacích externího charakteru, například vlastní mobilní aplikace, kde je možné celkem bezpečně uložit secret klienta,



Obrázek 5: Resource Owner Password Credentials Flow[22]

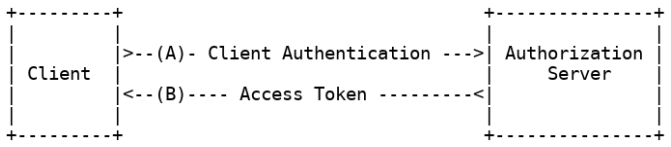
který je pak spolu s přihlašovacími údaji uživatele zasílán v požadavku pro získání přístupového tokenu, obrázek 5 krok (B). Tento grant spolu s krátkodobým přístupovým tokenem umožňuje získání i refresh tokenu, obrázek 5 krok (C), na rozdíl od Implicit Grantu, což je velmi výhodné ve prospěch uživatele. Pomocí refresh tokenu lze po vypršení přístupového tokenu, bez nutnosti uživatele zadávat znova přihlašovací údaje, získat nový přístupový token pro komunikaci. Refresh token má oproti přístupovému tokenu mnohem delší platnost a neměl by být použit pro komunikaci, nýbrž pouze pro obnovení přístupového tokenu. Bylo rozhodnuto, že tento grant bude podporován a využit v případné testovací mobilní aplikaci, kde se uživatel nebude muset při každém spuštění mobilní aplikace přihlašovat, ale při implementaci zohledním právě použití refresh tokenu tak, abych minimalizoval nutnou interakci uživatele.



Obrázek 6: Authorization Code Flow[22]

Pro aplikace s webovým rozhraním bylo nezbytné vytvořit podporu i pro přihlášení na daném systému s tím, že daná aplikace nebude mít přístup k přihlašovacím údajům uživatele, tak jako tomu bylo dosud, kdy každá aplikace řešila přístupové údaje a identitu uživatele po vlastní ose. Aplikace by v žádném případě neměla mít potřebu s těmito údaji nijak pracovat,

tohle bylo definováno jako kompetence autorizačního serveru nejen z důvodu zabezpečení, ale i chtěného sjednocení profilů uživatel. Authorization Code Grant je typ získání přístupu, který plně vyhovuje našemu požadavku. Při pokusu o přihlášení na webovou službu, obrázek 6 krok (A), dojde k okamžitému přesměrování na autorizační server s požadavkem na přihlášení, kde celý proces přihlašování je na straně AUTH serveru, jenž je plně kompetentní za validaci údajů uživatele, popřípadě změnu hesla či jeho obnovu v případě zapomenutí, obrázek 6 krok (B), a po úspěšném přihlášení dojde k přesměrování na původní službu, obrázek 6 krok (C). Přesměrování je provedeno na registrovanou redirect URI dané služby s vydaným dočasně platným kódem s životností maximálně v desítkách sekund. Aplikace potom na pozadí, obrázek 6 krok (D) a (E), provede výměnu tohoto dočasného kódu za přístupový token s povolením získání profilu uživatele a přihlásí uživatele bez nutnosti spravovat jeho přihlašovací údaje, ve své správě bude pouze uchovávat globální id uživatele získané z autorizačního serveru. Authorization Code Grant umožňuje bezpečné přihlášení uživatele do jiných aplikací MonkeyData bez zveřejnění přístupového tokenu a bude sloužit jako základní flow pro přihlašování uživatel a vytvoření služby SSO s tím, že se uživatel bude muset v rámci jedné relace přihlásit právě jedenkrát, ale bude moci toto přihlášení použít i na více než jen jedné webové aplikaci z rodiny MonkeyData.



Obrázek 7: Client Credentials Flow[22]

Pro interní systémy běžící jako služby na pozadí je bezpodmínečně nutné získat profil uživatele, se kterým se bude v danou chvíli pracovat, a to zejména pro přístup jménem uživatele k dalším aplikacím. Pro tuto interní komunikaci s ověřením přístupu, ale nelze použít žádnou z výše uvedených metod, protože u každé je potřeba, aby uživatel fyzicky souhlasil a tím povolil daný přístup. Ve specifikaci OAuth 2.0 je definován Client Credentials Grant, který umožňuje vydání přístupového tokenu bez přítomnosti uživatele, obrázek 7 krok (A) a (B), což je vhodné právě pro služby bežící na pozadí. Tyto systémy, například importní systém a zpracování dat, běží bez přerušení ve dne i v noci a bylo by nemožné zajistit kooperaci uživatele v moment, kdy by byl danou službou potřeba. U Client Credentials Grant je důraz na důvěrnost klienta, protože klient jako takový, respektive aplikace, jenž zastupuje, požaduje přístupový token, se kterým lze přistupovat ke zdrojům vlastníka a toto se děje automaticky bez jeho přítomnosti. I z tohoto důvodu se s přístupovým tokenem nevydává refresh token, který je zbytečný. Tento grant bude využit pouze pro ověřené interní aplikace bežící jako služby na zabezpečených serverech, kde je požadovaná diskrétnost zaručena.

Po zhodnocení všech typů přístupu, aktérů a cílů navrhované aplikace jsem vytvořil diagram tříd (viz příloha B, obrázek 14), kde jsou zahrnuty třídy reprezentující jednotlivé aktéry a také třídy nutné pro stěžejní funkce systému dle specifikace RFC6749. Z důvodu zapouzdření jsou

všechny atributy záměrně jako soukromé, při návrhu jsem vůbec neuvažoval o tom, že by nějaká třída umožňovala přímý přístup ke svému stavu bez použití metody k tomu určené. Záměr byl, aby každá třída měla jasně specifikovaný interface a tudíž byla i zaměnitelná a připravena pro možné budoucí rozšíření funkčnosti.

Základní entita systému je User, ta je s dalšími v kompozitním vztahu. Zde vycházím z požadavku, aby uživatel byl vlastníkem klientů, a proto i entita Client nemůže existovat bez User. Tohle se obrazí i pro entity AuthorizationCode a AccessToken, které se vážou k User a nemohou samy existovat. AccessToken je závislý i na Client, který požádal o vydání tokenu. Client nemusí mít žádná omezení, ale může mít omezení na povolení přístupů jen pro určité Scopes, kde pověření pro vydávaný přístupový token musí být jejich podmnožinou. Entita Client může mít i omezení pro povolené Grants. Entita Grant je zde navíc a nebyla by potřeba, ale rozhodl jsem se jí využít pro restrikci Client-Grant, kde lze omezit použití metody pro vydání přístupového tokenu a je možné toto udělat systémově. Lze pak vytvořit klienta s omezením například pouze na Resource Owner Password Credentials Grant pro použití v mobilní aplikaci rodiny MonkeyData a tento klient při zneužití jeho id a secret nebude moci provézt třeba Client Credentials Grant, kde není nutné mít přihlašovací údaje uživatele a tím jednoduše získat přístup k jeho zdrojům. Vztahem Client-Grant a jeho použitím zlepšujeme bezpečnost systému a ochranu dat uživatel. Entita RedirectUri je závislá na Client a nemůže existovat samostatně. Zvolil jsem možnost mít více RedirectUri i z důvodu testování, kde klient může běžet na staging prostředí a komunikovat s produkčním autorizačním serverem. Není pak potřeba mít několik klientů pro nutné finální testovaní aplikací před vydáním do produkce. Entita AuthorizationCode má právě jednu RedirectUri a seznam Scopes, které pak přebírá AccessToken při vytvoření. Entita RefreshToken je v kompozičním vztahu s AccessToken.

Připravil jsem datovou analýzu počínaje logickým modelem, který vychází z diagramu tříd (viz příloha B, obrázek 15) spolu s relačním modelem (viz příloha B, obrázek 16). Z uvedeného obrázku lze vidět, že krom primárních a cizích klíčů jsem zvolil i vhodné indexy pro zrychlení vyhledávání. Uživatele například budeme při přihlašovaní vždy vyhledávat pomocí emailu, ten musí být unikátní a také indexovaný.

Aplikace byla navržena jako třívrstvá, kde prezentační vrstva obsahuje vstupně výstupní kontrolery, byznys vrstva doménu aplikace a datová vrstva řeší přístup k úložišti (viz příloha B, obrázek 17). Na diagramu komponent lze vidět rozdělení aplikace, které koresponduje s částmi zmíněnými v kapitole 5.

6.3 Úkol č.2 - Implementace aplikace

6.3.1 Zadání úkolu

Další úkol, který jsem zpracovával, bylo implementovat jádro aplikace. To zahrnovalo převážně OAuth2 část aplikace, respektive její doménu, která měla rozšířit abstraktní doménový model a vytvořit základ pro komponenty z prezentační vrstvy.

6.3.2 Rozbor problému

Pro implementaci byl zvolen programovací jazyk PHP ve verzi 7.1, který zaručuje jak dostatečnou rychlost, tak i podporu OOP. Je to jazyk s delší historií, dostatečně rozšířený a se stabilní základnou. S ohledem na to, že tato aplikace by měla sloužit jako jakýsi středobod ekosystému dalších firemních aplikací a systémů, bylo nutné, aby byla kódová základna aplikace spravovatelná a lehce pochopitelná i ostatními členy týmu. Pro implementaci aplikace jsem zvážil otázku použití některého z dostupných frameworků, z použitelných připadaly v úvahu Nette[16], Symfony[25] a Laravel[14]. Ve firmě byla připravena upravená verze Laravelu, a i proto bylo zvoleno použití tohoto frameworku. Laravel vychází ze Symfony a přidává vlastní funkcionalitu. K té mimo jiné patří šablonovací systém Blade, propracovaný QueryBuilder a rozšířené routování. Laravel je i pro začátečníky dobrá volba, protože má rychlou učící křivku. Systém šablon Blade je jednoduchý, ale efektivní a na rozdíl od jiných šablonovacích systémů neomezuje použití čistého PHP v šablonách, což se dá považovat i za nevýhodu. Při běhu aplikace se šablona kompiluje jako PHP kód a nepřidává žádnou režii, dokud není změněna. Tohle chování se skvěle hodí např. při nastavené OPcache, kde šablony nejsou generovány při každém použití, ale při vhodném návrhu se celé načítají z paměti a při renderování se vkládají pouze proměnné. Routing v Laravelu je dost propracovaný a obsahuje spoustu metod, i proto je mnohdy dosti nepřehledný. V aplikaci je využit jen zčásti a to jako mapování URL na kontroler, kde na jednom místě je definována závislost, což se jeví jako velká výhoda pro případné řešení chyb. Zde je možné zařadit do zpracování požadavku tzv. middleware, které lze s výhodou použít pro řešení např. zabezpečení a kontroly přístupu. Toho se využívá především u API, kde každý endpoint je zařazen do skupiny pro middleware a zabezpečení se řeší na jednom místě a není nutné toto implementovat do každého kontroleru. QueryBuilder použijeme pro dynamické skládání SQL dotazů, zaručuje nám jejich správnou syntaxi a díky parametrizovaným dotazům a validaci zabraňuje SQL injection. V aplikaci není QueryBuilder použit přímo, ale je součástí datové vrstvy a je využíván abstraktní doménou.

6.3.3 Realizace

Z předchozího úkolu jsem měl připravený diagram tříd a na jeho základě jsem začal s implementací. Jako první jsem vytvořil interface `IUser` v namespace `Domain`, který rozšiřuje interface `Authenticable`, což je interface z frameworku Laravel, jenž nám zaručí použití objektu jako možné entity pro přihlášení. Nebude potřeba pak tuto funkcionalitu řešit zvlášť, ale můžeme použít logiku připravenou ve frameworku. Interface obsahuje metody pro přístup k atributům objektu a také metody (viz. níže), které jsou připraveny pro funkcionalitu přihlášení uživatele do aplikace.

```
namespace Monkey\Oauth2Server\Interfaces\Domain;

use Illuminate\Contracts\Auth\Authenticatable;
```

```
interface IUser extends Authenticatable {
   public function getEmail(): string;
   public function isSuspended(): bool;
   public function checkPasswordAgainstHash(string $password): bool;
...
}
```

Výpis 1: Interface IUser

Postupně jsem vytvořil i interface pro další entity počínaje `IClient`, `IScope`, `IGrant`, `IAccessToken` a `IRefreshToken`. Interface `IAccessToken` ale také `IRefreshToken` vycházejí z `IToken` a ten zase z `ICode`. Toto rozdělení jsem zvolil kvůli specializaci tokenů a také proto, že interface `IAuthorizationCode` dědí přímo z `ICode`, protože u něj metody `isRevoked()` a `revoke()` jsou zbytečné.

```
namespace Monkey\Oauth2Server\Interfaces\Domain;

interface ICode {
    public function isExpired(): bool;
}

interface IToken extends ICode {
    public function isRevoked(): bool;
    public function revoke();
}
```

Výpis 2: Interfaces ICode a IToken

Každá entita má kromě doménového interface a doménové třídy, také korespondující objekt model a objekt repositář. Model vychází z abstraktního modelu `AModel`, který na jednom místě řeší připojení k databázi a obsahuje finální metodu `getConnection()`, která nám vrací databázové připojení podle nastavení prostředí. V modelech díky abstraktnímu základu nemusíme kromě definice jména tabulky, nastavení časových atributů a příznaku pro smazání řešit vůbec nic jiného. Repositář také využívá model pro přístup do databáze a plní úlohu podle vzoru Table Data Gateway, uvnitř používá třídu `DomainMapper` podle vzoru Data Mapper, kde dochází k mapování databázových hodnot na konkrétní doménu. Vzhledem ke generičnosti názvů entit a modelů není třeba definovat názvy, jedině v případě, že se budou vymykat konvenci.

Hlavní třída naší domény je jednoznačně `User`, ten je potřeba pro funkčnost toho dalšího. Mimo obyčejné gettery a settery sloužící pro práci s objektem jsem vytvořil i pomocné metody pro práci s heslem tak, aby nastavení a hashování bylo nezávislé na frameworku, což bylo důležité pro bezpečnost. Je použit `Hash` s vlastní implementací založený na PHP funkci `password_hash`

využívající Bcrypt[5]. Už při standardním nastavení jsem díky adaptivnosti Bcrypt zvýšil počet iterací a tím zajistil lepší ochranu pro uživatelská hesla proti potenciálním útokům například pomocí Rainbow Tables.

```
namespace Monkey\Oauth2Server\Domain;

class User extends ADomain implements IUser {
...
    public function setPassword(string $password): IUser {
        $hashedPassword = $this->getHashInstance()->create($password);
        $this->hashedPassword = $hashedPassword;
        return $this;
   }

    public function checkPasswordAgainstHash(string $password): bool {
        if ($this->isPasswordSet() && $this->getHashInstance()->check($password
            , $this->getHashedPassword())) {
            return true;
        }
        return false;
    }

    private function getHashInstance(): Hash {
        return Hash::make(IUser::DEFAULT_ROUNDS_FOR_HASHING);
    }
...
}
```

Výpis 3: Část třídy User

Třída `User` také obsahuje metodu pro vytvoření klienta. Protože klient je závislý na uživateli, je kompetence uživatele vytvořit klienta. Metoda `createClient(string $name): IClient` přijímá jediný parametr a to je uživatelský název klienta, který je volitelný. Při vytvoření klienta je id a secret generovaný statickou metodou `createStringToken(int $tokenLength)` třídy `TokenCreator`, která generuje náhodný textový řetězec v zadané délce, pro id klienta je to 40 znaků a pro secret je to 100 znaků. Do vytvořené instance `Client` se předává instance `User` a vzniká závislost klienta na uživateli. Každý klient má svého vlastníka ve formě uživatele. Při běžném provozu ale `Client` může existovat samostatně, není potřeba, aby vlastník klienta byl přihlášen.

```
namespace Monkey\Oauth2Server\Domain;
```

```
class Client extends ADomain implements IClient {
...
    public function createAccessToken(IUser $user, int $tokenLength =
        IAccessToken::DEFAULT_TOKEN_LENGTH, IScopeHolder $requestedScopes = null
        ) {
        if ($requestedScopes === null) {
            $requestedScopes = new ScopeHolder();
        }

        $tokenCreator = new AccessTokenCreator($this, $requestedScopes);
        $accessToken = $tokenCreator->createAndGetAccessToken($user,
            $tokenLength);

        return $accessToken;
    }

...
}
```

Výpis 4: Část třídy Client

Třída `Client` je zodpovědná za vytváření autorizačních kódů a přístupových tokenů. Využívá k tomu `AuthorizationCodeCreator` respektive `AccessTokenCreator`, obě dvě třídy dědí z `ATokenCreator` a přidávají další závislosti nutné pro vznik tokenů. U autorizačního kódu je to objekt `RedirectUri`, který v sobě nese informace o konkrétní redirect URI a klientovi, pod kterého patří. PHP sice obsahuje metody a objekty pro parsovaní URI, ale práce s nimi není ucelená. Připravil jsem proto `UrlParser`, který obaluje více nastrojů a přidává k nim i vlastní funkcionalitu pro jednoduchou práci s URI. Třída `UrlParser` a její funkce jsou využity ve třídě `RedirectUri`.

Po vytvoření všech objektů domény dle návrhu, tj. modelů, relací, repositářů, a pomocných tříd především pro vytváření kódů a tokenů, jsem prováděl testování objektů s využitím PHPUnit frameworku. Vývoj a programování zčásti probíhalo jako Test Driven, kdy například u objektu `RedirectUri` jsem prvně vytvořil testovací třídu a pole korektních a nekorektních URI, které se testovaly, a posléze řešil vlastní implementaci validace redirect URI, na kterou byly kladeny specifické nároky na strukturu. I díky tomuto přístupu se podařilo odladit většinu chyb již při vývoji.

Doména jako taková nám řeší přístup k databázi a také vztahy a závislosti mezi entitami. Tyto služby především využívá objekt `Server`, který je součástí balíčku a to proto, že vytváří hlavní vstupní bod pro celou logiku a zastřešuje funkcionalitu části OAuth2, která bude poté

využita ve vstupně výstupních kontrolerech na prezentační vrstvě. Třída `Server` implementuje interface `IServer` a stěžejní funkce pracují s interface `IGrant` a `IResponse`.

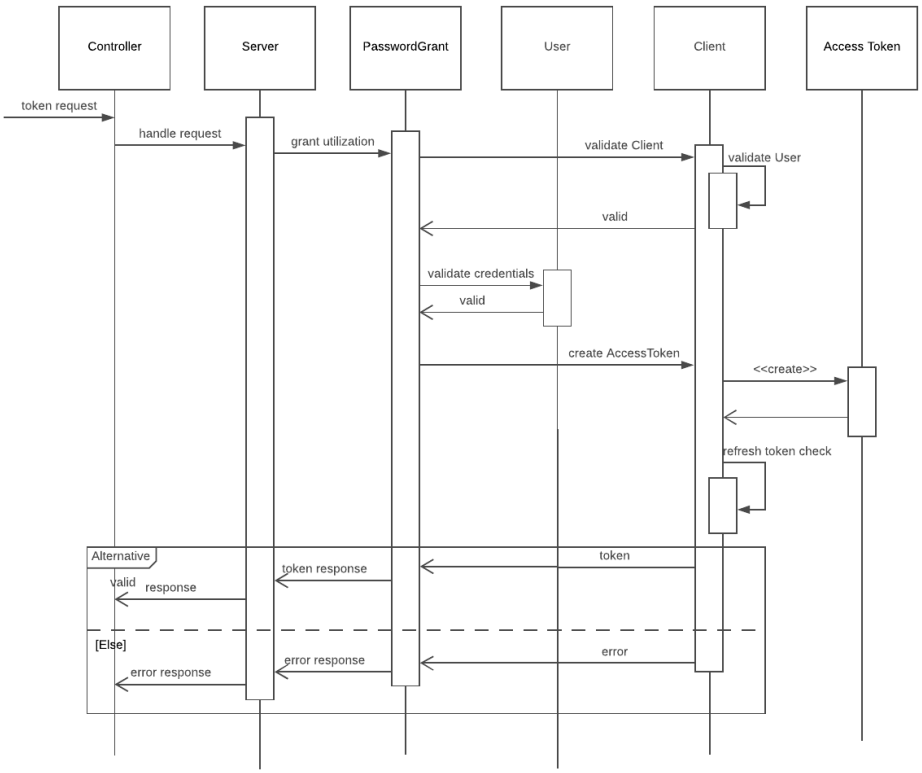
```
namespace Monkey\Oauth2Server\Interfaces;

interface IServer {
    public function addGrant(IGrant $grant);
    public function addResponseType(IResponseType $responseType);
    public function validateRequest(): bool;
    public function getResponse(): IResponse;
    public function setAuthenticatedUser(IUser $user);
    public function createTemplateManager(string $templateManagerClassName,
        string $screenCode = null): ITemplateManager;
}
```

Výpis 5: Interface IServer

Pomocí metody `addGrant(IGrant $grant)` je možné nastavit grant pro zpracování. Interface `IGrant` je základní pro skupinu tříd implementujících jednotlivé formy získání přístupového tokenu. Díky tomuto návrhu je možné, aby jeden vstupní bod mohl reagovat na více než jen jeden grant. To je důležité například pro endpoint řešící vydání tokenů. Kontroler inicializuje `Server` a přidá jednotlivé typy grantů, které je schopen zpracovat. To, že je `Server` schopen reagovat na určitý grant, neznamená, že každý klient může požádat o přístupový token. V jednotlivých třídách je striktní logika kontrolující a validující všechny potřebné parametry a prerekvizity nutné pro úspěšné vydání přístupového tokenu. Ta vychází z abstraktní třídy `AGrant`, která řeší společné prvky a je postupně rozšířena v konkrétních třídách. Implementované typy grantů jsou `AuthorizationCodeGrant`, `PasswordGrant` a `ClientCredentialsGrant`.

Třída `Server` obsahuje metodu `validateRequest()`, která může být po inicializaci a nastavení objektu volána kontrolerem pro lepší ovládání. Jak metoda `validateRequest()`, tak `getResponse()` by měly být v kontroleru obklopeny konstrukcí try-catch, a to proto, že při zpracování `Server` vyhazuje velké množství vyjímek, které jsou reakcí na chyby vzniklé při zpracovávání grantů. Vyjímek je více než deset a každá je reakcí na jinou chybu anebo problém vzniklý při zpracování, všechny dědí z abstraktní třídy vyjímek `AOauth2ServerException`. Výjimky, mimo jiné `AccessDeniedException`, `UnauthorizedClientException`, `RedirectUriException`, se společným předkem se základními metodami je možné využít v kontroleru, kde by se měly odchytávat a kontroler by měl odekvátně reagovat. Vytvořil jsem i pomocnou třídu `Oauth2Error` se závislostí na `AOauth2ServerException`, která normalizuje chybové odpovědi podle specifikace OAuth 2.0 a funguje jako prostředník, jenž lze použít v kontroleru pro vytvoření chybové odpovědi pomocí třídy `ErrorResponse`. Třída `Oauth2Error` skutečně kontroluje a potlačuje jiné vyloženě serverové či aplikační chyby tak, aby při formulování chybové odpovědi nedošlo v žádném případě k úniku citlivých údajů, se kterými se pracuje. Je v kompetenci kontroleru



Obrázek 8: Sekvenční diagram PasswordGrant

nestandardní chyby zaznamenat pomocí nástrojů tomu určených. Po úspěšném zpracování požadavku `Server` prostřednictvím metody `getResponse()` vrací objekt s odpovědí implementující `IResponse`. Předání této odpovědi je již v kompetenci kontroleru na prezentační vrstvě.

6.4 Úkol č.3 - Výběr úložiště

6.4.1 Zadání úkolu

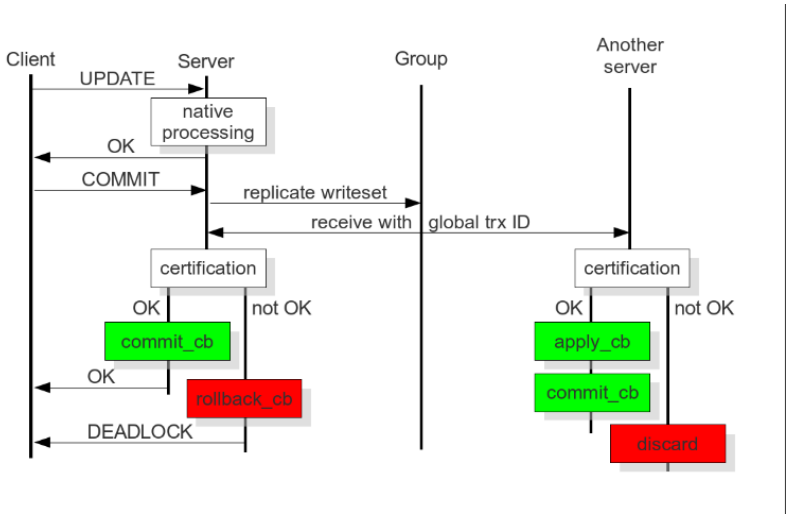
Mým další úkolem bylo vyzkoušet a zvolit technologii úložiště. Pracovalo se s open-source softwarem MySQL[15] a jeho klony. Cílem bylo vybrat nejvhodnější kombinaci replikace pro splnění požadavku vysoké dostupnosti. Výstupem tohoto úkolu mělo být jednak zvolení technologie po konzultaci s vedením a také jeho fyzický test.

6.4.2 Rozbor problému

V úvahu jsem bral hlavní cíl systému, vytvoření vysoce dostupné služby, která poběží ve více instancích a pro zajištění provozu potřebuje i vysoce dostupné úložiště. Bylo nutné se seznámit s jednotlivými druhy replikace, které MySQL nabízí, a také s různými verzemi vyvíjenými několika společnostmi.

6.4.3 Realizace

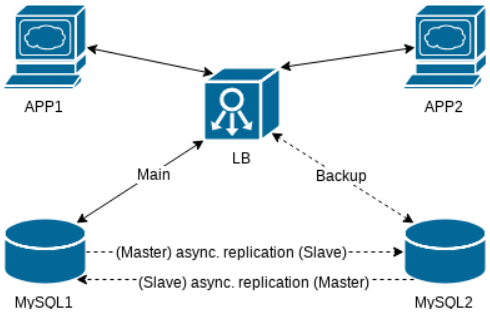
Při řešení tohoto úkolu jsem začal sběrem informací o MySQL a replikaci obecně. Zaměřil jsem se jen na novější verze MySQL, konkrétně verzi 5.7. Podporované typy replikace jsou dva, je to asynchronní replikace a semi-synchronní replikace. Asynchronní replikace je starší, podporovaná i v dřívějších verzích, braná jako defaultní. Při této replikaci master zapisuje události do binary logu a slave je postupně čte. Tato replikace je někdy nazývaná jako volná, protože master nečeká na slave, který při čtení binary logu může být pozadu a jeho aktuální stav nemusí být totožný s master serverem. Vzniká zpoždění při replikaci, které u zatížených systémů může být i několik sekund, ale výjimkou není ani větší zpoždění. Výhodou je, že při zápisu na master dochází jen k minimálnímu zpoždění, protože se nečeká na potvrzení zápisu slavem. Tato replikace je vhodná především pro aplikace, kde potřebujeme mít více replik pro čtení a nevadí nám zpoždění při replikaci, lze také použít jako online zálohu databáze pro případ výpadku master serveru. Semi-synchronní replikace naproti tomu požaduje potvrzení od alespoň jednoho dalšího serveru (obrázek 9) a v případě selhání je chyba propagována směrem k aplikaci. Aplikace se tedy musí rozhodnout, jak s chybou naložit a zda opakovat zápis. Semi-synchronní replikace nabízí vyšší dostupnost beze ztráty dat, avšak je oproti asynchronní replikaci pomalejší na zápis, což může být problém u systémů s větším počtem zápisů než čtení. Novější verze MySQL přicházejí i s GTID, což je unikátní id transakce, které je nejen jedinečné v rámci jednoho serveru, ale je jedinečné i ve skupině serverů. Globální id transakce pomáhá ke zlepšení robustnosti replikace, a to jak asynchronní, tak i semi-synchronní.



Obrázek 9: Multi Master Replication[18]

V našem systému požadujeme, aby aplikace měla co nejaktuálnější data a také aby v případě výpadku či restartu jednoho databázového serveru služba běžela bez přerušení. Po debatě s kolegy jsem proto ihned vyřadil replikaci master-slave, která se spíše hodila pro datové sklady, analytiku či online zálohu. Databázové servery pro aplikaci měly být rovnocenné a navzájem

zastupitelné, což v případě master-slave není zaručeno. Během výběru topologie jsem zvažoval i klasický MySQL server s nastavením replikace master-master (viz obrázek 10), což je nastavení, kdy každá z instancí je jak master (ze svého pohledu), tak i slave (z pohledu druhé instance). Toto nastavení by zaručilo jak vysokou dostupnost, tak i zastupitelnost databázových strojů, poněvadž při výpadku by funkci zastoupil druhý master server. Při tomto řešení by ale každá instance aplikace musela vědět o obou databázových serverech a řešit vnitřně nějaký druh load balancingu. To by trochu zvyšovalo komplexitu aplikace a aplikace by i řešila něco, co by nemělo být v její kompetenci. Proto jsem před databázové servery zařadil L4 load balancer, který by posílal komunikaci z aplikace bežící ve více instancích (APP1 a APP2) na první databázový server. Pro aplikaci by se tak databázová vrstva, respektive její vstupní bod tvořený load balancerem, jevila jako jeden server. Při výpadku MySQL1 by load balancer automaticky přepnul na Backup a posílal komunikaci na MySQL2. Toto řešení se zdálo býti jednoduché a efektivní, plnilo očekávanou zastupitelnost a vzhledem k tomu, že zde není použita semi-synchronní replikace, tak i zápisy by nebyly zpomalovány nutností potvrdit transakci druhým serverem.



Obrázek 10: MySQL master-master replikace

Po domluvě s vedoucím jsem dostal možnost tuto topologii zkusit v praxi. Na dva testovací počítače jsem nainstaloval MySQL verze 5.7 a prvnímu přiřadil `server-id 101`, na druhý jsem nastavil `server-id 102`. V produkčním prostředí bychom použili nezávislý load balancer, abychom zajistili potřebnou paralelizaci. V případě testů jsme jako load balancer využili HAProxy nainstalované na první testovací počítač s tímto nastavením.

```
...
listen auth-mysql
    bind 0.0.0.0:3307
    mode tcp
    server mysql1 10.10.1.101:3306 check
    server mysql2 10.10.1.102:3306 check backup
...
```

Výpis 6: HAProxy nastavení Master-Master

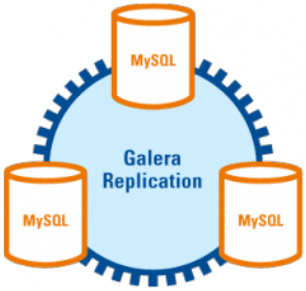
Po nastavení replikace MySQL ze serveru 101 na 102 a obráceně jsem mohl toto zapojení zkusit přímo na aplikaci. Jednoduchý testovací scénář se skládal z:

1. test zápisu a čtení z aplikace, vypnutí služby MySQL1
2. test zápisu a čtení z aplikace, reset služby MySQL1
3. vydání autorizačního kódu, vypnutí služby MySQL1, vydání přístupového tokenu
4. vydání autorizačního kódu, reset služby MySQL1, vydání přístupového tokenu

Již při první scénáři se ukazovalo, že největší problém a velmi velkou nevýhodou tohoto řešení je asynchronní replikace u obou zmíněných databázových systémů způsobující poměrně velké zpoždění přenosu dat na druhý server. Data byla replikována obousměrně, ale ne dostatečně rychle. Zápis na MySQL1 probíhal velmi rychle a nebyl žadný problém ani se čtením, load balancer vše korektně směroval na MySQL1. V případě vypnutí prvního serveru se ale i přes korektní ukončení služby projevilo zpoždění asynchronní replikace, na server MySQL2 se nestačila replikovat všechna data včas. Přistoupil jsem ke scénáři č. 2, kde se problém prohloubil. Při resetu instance databáze se celý systém MySQL1 okamžitě vypnul, žádná další data se nestačila replikovat na druhý server a vznikla situace, kdy na prvním serveru byla data uložena, ale neexistovala na serveru druhém, což způsobovalo chyby v aplikaci, která korektně zapsala, ale pak nemohla data přečíst. Zpoždění asynchronní replikace při neustálém zápisu se pohybovalo v řádech vyšších desítek až stovek milisekund, při resetu byl load balancer schopný přepnout z nefunkční instance a přesměrovat komunikaci během několika milisekund, tím vzniklo „datové okno“ o velikosti desítek až stovek milisekund. Testovací scénář č. 3 a 4 potvrdily závažné nedostatky tohoto řešení. Nasimuloval jsem situaci, kdy klient žádá uživatele o povolení přístupu, uživatel povolení schvaluje a autorizační server vydává autorizační kód, který se posílá klientovi, ten pak s tímto kódem a svými údaji žádá o vydání přístupového tokenu jménem vlastníka zdrojů. Pokud proces selže, je třeba jej opakovat celý znova. Během tohoto procesu se v případě scénáře č. 3 provádělo vypnutí první databázové instance. V tomto případě došlo k jednomu selhání vydání přístupového tokenu na každých šest pokusů. U scénáře č. 4, kdy došlo k resetu instance, byl počet selhání skoro dvojnásobný než u scénáře č. 3. Po konzultaci s vedoucím bylo rozhodnuto, že tato topologie a replikace master-master není pro autorizační server vhodná z důvodu nespolehlivosti při výpadku prvního databázového serveru. Nutno ještě podotknout, že v případě resetu MySQL1 pak instance startovala delší dobu, ale pro load balancer se jevila jako aktivní a plně použitelná, což způsobovalo další nekonzistenci aplikačních dat.

Na doporučení vedoucího jsem do testu zařadil řešení XtraDB Cluster od firmy Percona[18], které využívá klasických MySQL serverů a přidává k nim tzv. Galera Replication (viz obrázek 11), což je semi-synchronní multi-master replikace, která nabízí vícevláknovou replikaci, nulové zpoždění replikace a transparentnost pro aplikace, a to vše s důrazem na konzistenci dat. Data jsou potvrzena a zapsána buď na každou instance, anebo nikde. Výhodou je, že na každou

instanci clusteru může být zapisováno, každopádně doporučení je zapisovat na jeden server s tím, že v případě výpadku se zapisuje na další v pořadí.



Obrázek 11: Galera replikace[18]

Jednou z vlastností XtraDB je možnost zvýšení (anebo i snížení) počtu instancí za běhu bez výpadku služby se zajištěním požadované vysoké dostupnosti. Nutno zmínit fakt, že XtraDB Cluster není vhodný pro aplikace s větším počtem zápisů než čtení z důvodu výše zmíněné semi-synchronní replikace, kde zápis je nepatrně brzděn potvrzením dalších instancí. V naší situaci se však předpokládá rozložení zápis a čtení v poměru 20/80.

Přistoupil jsem k otestování XtraDB. V dokumentaci jsem si prošel instalaci, která se v podstatě neliší od klasického MySQL serveru. Jiné je ale nastavení a počáteční práce s clusterem. Aby vše správně fungovalo, je důležité mít ve skupině XtraDB alespoň 3 servery. Replikace dat v clusteru není volná a jednotlivé stroje vytvářejí kvorum, které pro správnou funkci clusteru musí mít nadpoloviční většinu z důvodu, aby nedošlo k situaci tzv. „split brain“. Split brain situace může nastat u dvou serverů například při přerušení komunikace mezi nimi, kdy žádný z nich není schopen určit, zda má být aktivní či ne. Využil jsem stávající testovací počítače, přidal k nim třetí a na každý z nich nainstaloval percona-xtradb-cluster-57, který koresponduje s verzí MySQL 5.7. Zde, narozdíl od klasické replikace, která se nastavuje v konzoli mysql, bylo potřeba toto nastavení přidat do konfiguračních souborů na každý server do sekce `[mysqld]`. V nastavení všech byla společná `wsrep_cluster_address`, která definuje základní skupinu serverů patřících do XtraDB clusteru. V konfiguraci jsem nastavil i korespondující `server-id` a `wsrep_node_address`.

```
[mysqld]
server-id = 101
wsrep_provider=/usr/lib/libgalera_smm.so
wsrep_cluster_address=gcomm://10.10.1.101,10.10.1.102,10.10.1.103
wsrep_node_address=10.10.1.101
```

Výpis 7: XtraDB Galera konfigurace

Pro spuštění clusteru je potřeba spustit službu na prvním serveru bez nastaveného parametru `wsrep_cluster_address` a až pak spustit další. Abych ale nemusel konfiguraci měnit, provedl

jsem bootstrap clusteru pomocí příkazu `/etc/init.d/mysql bootstrap-pxc`, kde se ignoruje nastavení v konfiguraci a tento node předpokládá, že je první spuštěný a inicializuje cluster jako takový. Další node je možné spustit už standardním příkazem, automaticky se podle nastavení připojí na další servery definované v konfiguraci. Topologie se nám oproti předchozí rozšířila (příloha C obrázek 18) a bylo nutné i upravit konfiguraci HAProxy tak, aby korespondovala s doporučeními a současným stavem.

```
...
listen auth-mysql-read
    bind 0.0.0.0:3307
    mode tcp
    option httpchk
    server mysql1 10.10.1.101:3306 check port 9200
    server mysql2 10.10.1.102:3306 check port 9200
    server mysql3 10.10.1.103:3306 check port 9200

listen auth-mysql-write
    bind 0.0.0.0:3308
    mode tcp
    option httpchk
    server mysql1 10.10.1.101:3306 check port 9200
    server mysql2 10.10.1.102:3306 check port 9200 backup
    server mysql3 10.10.1.103:3306 check port 9200 backup
...
```

Výpis 8: HAProxy nastavení XtraDB

Po inicializaci všech tří instancí jsem přistoupil ke stejnému testu, jaký jsem prováděl na master-master topologii. Pro změření rychlosti replikace jsem ještě navíc nainstaloval a nastavil Percona Monitoring a Management tak, abych mohl kontrolovat a monitorovat chování clusteru. Cluster si ve scénářích č. 1 a 2 vedl podobně, z pohledu aplikace nebylo poznat, zda databázový server byl vypnut anebo resetován. Vypínal anebo resetoval jsem vždy jen první ze tří serverů, aby kvorum bylo nad padesáti procenty. Pokud bychom chtěli cluster, kde by bylo možné ztratit dva servery v jeden moment, museli bychom mít velikost clusteru 5. Při testování scénářů č. 3 a 4 jsem nepozoroval velký počet selhání, vydání autorizačního kódu a následná výměna za přístupový token se nepovedla jen ve dvou případech při resetu instance, které z celkového počtu tvořily méně než jedno procento. Ještě bych podotknnul, že při vypnutí první instance se velikost clusteru snížila na 2 a cluster takto fungoval dále, při startu se vypnutá instance připojila zpět a před zviditelněním pro load balancer se provedla potřebna synchronizace dat, teprve poté byla instance zpřístupněna. V případě resetu, zůstala velikost clusteru 3 a proces zařazení resetované instance do clusteru proběhlo stejně jako u instance vypnuté. Synchronizace

ve všech případech trvala jen několik sekund. Zpoždění replikace v XtraDB clusteru je oproti asynchronní menší o 2 a více řádů, viz. obrázek 12. Jak je na grafu níže vidět, rychlost replikace se na jednotlivých serverech pohybovala v průměru pod jednu milisekundu.

Obrázek 12: Průměrná doba replikace XtraDB

Po konzultaci s týmem na základě uvedených testů bylo rozhodnuto jako úložiště pro aplikaci použít řešení XtraDB Cluster od firmy Percona, které zaručuje požadovanou vysokou dostupnost, rychlost i velkou míru stability.

6.5 Úkol č.4 - Mobilní aplikace

6.5.1 Zadání úkolu

Jelikož dnešní doba je ve jménu mobility, mým dalším úkolem bylo navrhnout a naprogramovat mobilní aplikaci pro platformu Android[1] s podporou API level 19 (KitKat 4.4) a vyšší. Cílem tohoto úkolu bylo vytvořit jednoduchou mobilní aplikaci komunikující s webovým API a používající Resource Owner Password Credentials Grant pro přihlášení uživatele a získání přístupového tokenu.

6.5.2 Rozbor problému

Na realizaci tohoto úkolu jsem byl částečně připraven a měl nainstalované potřebné nástroje pro vývoj. Bylo nutné tedy jen si osvěžit znalosti o platformě Android z dokumentace a připravit návrh pro jednoduchou aplikaci, která bude sloužit jako prototyp pro další vývoj.

6.5.3 Realizace

V případě mobilní aplikace jsem začínal od úplného začátku, neměl jsem k dispozici žádnou adresářovou strukturu ani vytvořené balíčky, které bych mohl využít. Začal jsem tedy návrhem jednotlivých částí, ze kterých jsem měl v plánu aplikaci poskládat, a složky vytvořil v moment, kdy jsem vytvářel třídy podle typu použití. Inspiroval jsem se v mé předešlé mobilní aplikaci a nakreslil si mapu, kde jsem si napsal jednotlivé části, třídy a funkce. Aktivity jsem logicky rozdělil na tři části, před přihlášením, přihlášený uživatel a ostatní. Začal jsem projekt v Android Studiu, které za mě část práce udělalo a vytvořilo XML soubory s nastavením. Vytvořil jsem základní aktivitu aplikace a nazval ji `LoginActivity`, ta řeší přihlášení uživatele. Tato aktivita jako jediná dědí přímo z `AppCompatActivity`. Pro část aplikace, kdy je uživatel již přihlášen, jsem vytvořil

abstraktní aktivitu `ADrawerActivity`, která také dědí z `AppCompatActivity` a řeší společnou funkčnost stejnou pro všechny aktivity v přihlášené části. Společné je využití vyjíždějícího menu (viz příloha D obrázek 19), funkčnost horního menu, listenery na tlačítka a samozřejmě obsluha přihlášeného uživatele mobilní aplikace s použitím objektu `UserProvider` a modelu `User`. Výjimka z tříd aktivit je `AboutActivity`, která stojí úplně samostatně, ale neřeší žádnou logiku, jen zobrazuje informace o autorovi.

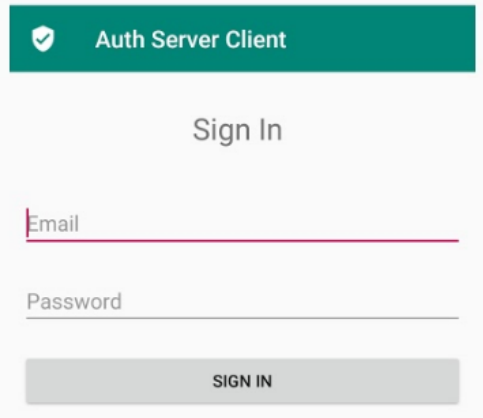
```
public interface ApiInterface {
    @POST("o/oauth2/token")
    Call<Token> getTokenByPasswordGrant(
            @Query("grant_type") String grantType,
            @Query("client_id") String clientId,
            @Query("client_secret") String clientSecret,
            @Query("state") String state,
            @Query("email") String email,
            @Query("password") String password
    );

    @GET("api/user/overview")
    Call<Response<Overview>> getUserOverview(
            @Query("access_token") String accessToken
    );
...
}
```

Výpis 9: ApiInterface

Po konzultaci s kolegy z mobilního vývoje jsem vytvořil interface `ApiInterface` (Výpis 9), který definuje request a response strukturu pro API volání. Interface staví na externím balíčku Retrofit2[21] a dost podstatně zjednodušuje práci s API a objekty, které pak přebírám k dalšímu zpracování. Třída `ApiClient` pak zastřešuje nastavení komunikace, základních hlaviček a automatické přibalení přístupového tokenu, který je nutný pro úspěšné získání odpovědi ze serveru. Zde také probíhá kódování požadavku a dekódování odpovědi pomocí třídy `GsonHelper`, která využívá nosný objekt `JsonString` a `JsonStringDeserializer` pro bezproblémový převod. Všechny tyhle pomocné třídy využívají třídu `Gson`, která je od společnosti Google a pracuje jako převodník pro formát JSON, který je použit v komunikaci. S třídou `ApiClient` a třídou `ConnectionHelper`, která zprostředkovává pomocné funkce od třídy `ConnectivityManager`, jsem byl nyní schopen s minimálním kódem v aktivitách provádět ošetřená a jednotná volání na vzdálený server. Výstupem mi vždy byly požadované datové objekty anebo chyba, která byla buď očekávaná třeba v případě neplatných údajů a pomocí třídy `MessageHelper` využívající

`Toast` zobrazena uživateli, anebo neočekávaná, kde se zobrazila standardní zpráva o chybě při komunikaci.



Obrázek 13: Přihlášení uživatele v mobilní aplikaci

Při startu aktivity `LoginActivity` se zobrazuje visuál (obrázek 13) skládající se z polí pro jméno a heslo. V aktivitě je vytvořeno několik funkcí, které řeší zahájení asynchronní komunikace, kde se uživateli zobrazí točící se loader, a s využitím třídy `ApiClient` se vrací jako výsledek dotazu objekt `Token` obsahující platný přístupový token pro uživatele. Token se ukládá a startují se služba `MediaPlayerService` a aktivita `OverviewActivity`, která je jako hlavní vstupní bod do části přihlášeného uživatele (viz příloha D obrázek 20). Služba `MediaPlayerService` nedělá nic jiného, než že přehraje zvuk po přihlášení a ukončí se. Uložení tokenu provede třída `TokenProvider` připravená pro obsluhu a práci s tokenem. Ta využívá třídu `SharedPreferencesHelper`, jenž využívá společnou paměť aplikace pro nastavení a usnadňuje práci zejména tím, že je schopna serializovat celé objekty a ukládat je jako JSON string. Ve svých útrobách využívá třídu `SharedPreferences`.

```
public class OverviewActivity extends ADrawerActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        FragmentTransaction fragmentTransaction = getSupportFragmentManager().
            beginTransaction();
        Fragment fragment = new OverviewFragment();
        fragmentTransaction.replace(R.id.content_frame, fragment);
        fragmentTransaction.commit();
    }
}
```

Výpis 10: OverviewActivity

Aktivita `OverviewActivity` a všechny další aktivity z části aplikace, kdy je uživatel přihlášen a které dědí z `ADrawerActivity`, nedělají nic jiného než že připravují korespondující fragment, například `OverviewFragment`, který se stará o získání potřebných dat a vykreslení obrazovky. Vykreslený fragment se vkládá do `R.id.content_frame`, který je z počátku prázdný a je součástí základního layoutu ovládaného abstraktní třídou `ADrawerActivity`. Sjednoceným základním rozložením společných prvků aplikace jsem docílil stejného vzhledu a konzistentního ovládání v části, kdy je uživatel přihlášen. Každý fragment ve svém kontextu samozřejmě může mít vlastní funkcionalitu navázanou na ovládací prvky vložené ve fragmentu. Toho využívám v aktivitě `ClientsActivity` ovládající fragment `ClientsFragment`, kde je zobrazen list vlastních klientů uživatele a každý klient je prokliknutelný na detail. Pro tuto funkčnost jsem vytvořil `ClientsAdapter`, který ve fragmentu `ClientsFragment` získává pomocí třídy `ApiClient` nosný objekt `Clients` držící pole objektů typu `Client`, to jest jednotlivých klientů, a nastavuje `ClickListener` na každou položku seznamu. Při prokliku se vyvolává aktivita `ClientActivity`, která zobrazuje a obsluhuje konkrétní záznam klienta.

Kromě aktivity `LogoutActivity` jsem vytvořil a připravil i aktivitu a fragment s komunikací pro ovládání vlastněných přístupových tokenů a sekci pro nastavení aplikace. Díky `ApiInterface` je velmi jednoduché přidat další volání a odpověď v kombinaci s vhodným datovým objektem. Mobilní aplikace zatím nemá žádné rozšířené funkce, ale splňuje základní požadavky úkolu a umožňuje pohodlné rozšiřování funkcionality při dalším vývoji.

6.6 Úkol č.5 - Kontejnerizace aplikace

6.6.1 Zadání úkolu

Jeden z nejvíce vyzývajících úkolů pro mne bylo zadání dát aplikaci do Dockeru[8]. Cílem bylo vytvořit Docker image, kde bude aplikace a její systémové závislosti, a to ve stavu, kdy tento image bude spustitelný ve formě kontejneru na platformě Kubernetes[13].

6.6.2 Rozbor problému

Vytvoření Docker image vyžaduje mít v prvé řadě nainstalováný Docker engine. Samotné vytvoření image je provedeno na základě předpisu, který se nazývá Dockerfile. Před vytvořením tohoto Dockerfile bylo potřeba si projít dokumentaci a seznámit se s jednotlivými příkazy, kterými se Docker ovládá, a možnostmi nastavení. Vzhledem k mé malé zkušenosti s tímto nástrojem mě čekalo hlavně učení, jak to všechno vlastně funguje.

6.6.3 Realizace

Nejdřívě jsem potřeboval nainstalovat a zprovoznit Docker na svém počítači tak, abych se s tímto nástrojem mohl blíže seznámit. Po přečtení dokumentace a zvláště sekcí jak to funguje, koncepty a jak začít jsem si našel možné způsoby instalace pro můj systém a po stažení cca 80MB in-

stalačních dat, jsem již mohl zkusit spustit příkaz `docker run hello-world`, který mi zprvu nahlásil, že nemůže najít image, ale posléze došlo ke stažení z registru a spuštění hlásící úspěšnou instalaci. Začal jsem tedy s přípravami na kontejnerizaci aplikace. Seznámen se základními pojmy jsem, dle ověřených postupů a příkladů, rozdělil kontejnerizaci na několik předpisů. Jako první jsem vytvořil Dockerfile pro základní balíčky, pomocné programy a knihovny postavené na systému Ubuntu[26] Server 16.04 LTS. Zde jsem cílil na vytvoření image, jenž bude pomocí docker příkazu spustitelné a bude se chovat obdobně jako virtuální mašina, se kterou jsem se setkal již dříve. Postupoval jsem po iteracích a nesnažil se udělat vše najednou. Po několika málo buildech se mi podařilo na základě předpisu (viz Výpis 11) vytvořit image, který jsem použil jak na bližší seznámení s konceptem a funkčností, tak i jako základní obraz pro kontejnerizaci aplikace, byť tento image obsahuje i knihovny, které pro běh aplikace nejsou potřeba a jsou tam tedy navíc. Při finálním buildu bychom tedy mohli i tento základní obraz ještě zeštíhlit.

```
FROM ubuntu:16.04
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update \
&& apt-get -y --no-install-recommends install software-properties-common python
    -software-properties htop mc apt-utils wget jq unzip stress nmap iputils-
    ping \
&& apt-get update && apt-get clean
```

Výpis 11: Předpis základního obrazu mdUbuntu

Každý obraz má generovaný identifikátor a lze jej pojmenovat. To je důležité pro další práci s takovým obrazem. Zjednodušeně řečeno, Docker funguje na principu vrstev. Obraz, který jsem si připravil, může sloužit jako základní pro nějaký další. Mám tam připraveno vše, co v základním obrazu mám k dispozici a mohu to buď využít anebo překrýt. Vytvořil jsem ještě jeden obraz (image `mdPHP`, příloha E Výpis 13), který využívá mnou již vytvořený `mdUbuntu` a který obsahuje PHP knihovny a rozšiřující moduly pro PHP včetně nástroje Composer[7], který slouží pro správu balíčků pro PHP.

Vždy pouze poslední vrsta je schopna čtení i zápisu, všechny vrstvy pod ní je možné pouze číst. Hlavní výhoda rozdělení na několik předpisů a tím i vrstev je ta, že pro testování drobných změn v aplikaci, která je zpravidla na poslední vrstvě, nepotřebuji při každé změně vytvářet jeden obrovský dlouho trvající build. Toto mi dopomohlo k tomu, že jsem ušetřil dost času, kdy bych jen čekal na dokončení vytvoření obrazu aplikace, abych ji posléze mohl vyzkoušet spustit v kontejneru. Všechny předchozí obrazy již není potřeba při použití znovu vytvářet, a tak je poslední aplikační předpis dost štíhlý a finální build aplikačního image je velmi rychlý, což je obrovská výhoda právě pro testování.

Aplikaci bylo potřeba již „jen nahrát“ do kontejneru a spustit. V této části jsem ovšem narazil na problém. Při definici Dockerfile je nutné uvést tzv. entrypoint, kterým se definuje příkaz z image spuštěný jako proces číslo 1. Tento proces je zodpovědný za inicializaci aplikace. Pokud

tento proces skončí svůj životní cyklus, běh kontejneru se zastaví. V případě aplikace postavené například na programovacím jazyku Java bychom jako výsledek buildu měli `jar` soubor a ten bychom mohli použít jako entrypoint definovaný třeba takto:
`ENTRYPOINT ["java -jar /data/source/app.jar"]`

V PHP je tomu trochu jinak. Rozhodnul jsem se proto vytvořit vlastní inicializační soubor `md-init`, příloha E Výpis 14, a využít funkci frameworku Laravel. Ve výsledném Dockerfile souboru pro build aplikace jsem definoval adresář aplikace v kontejneru `/data/source/`, zpřístupněný port 8000 a jako entrypoint nově vytvořený soubor `/data/source/md-init`. Adresář aplikace v kontejneru `/data/source/` je mnou určený a nemá žádný speciální význam.

```
FROM mdPHP
COPY . /data/source/
WORKDIR /data/source/
STOPSIGNAL SIGINT
EXPOSE 8000
ENTRYPOINT ["/data/source/md-init"]
```

Výpis 12: Předpis obrazu aplikace

V init skriptu `md-init` jsem pro úspěšný start aplikace v testovacím módu potřeboval připravit několik proměnných, hlavně šlo o nastavení PHP limitu paměti, a pak i pomocné funkce pro výpis použité paměti kontejneru a základní funkci pro odchycení chyby při pádu aplikace a reakce na ní. Pro nastavení limitu paměti PHP jsem využil systémové cgroup, na kterých vlastně staví i Docker, a pro PHP jsem nastavil paměť o 10M menší než bylo pro kontejner. Tato hodnota se během zkoušení ukázala být jako dostatečná rezerva pro bezproblémový běh kontejneru. Na konci `md-init` je pak spuštěn skript z frameworku Laravel, který vytvoří virtuální webový server podle našeho nastavení na portu 8000.

Aplikaci se podařilo s úspěchem dokerizovat a je připravena na testovací spuštění v Kubernetes. Vestavěný webový server není sice tak výkonný jako například Apache[3] nebo Nginx[17], ale pro tento úkol je dostatečný. Díky pomocnému init skriptu je automaticky nastaveno prostředí podle vlastností kontejneru a zobrazen výpis a propagována chyba aplikace i v případě signálu SIGKILL.

7 Hodnocení znalostí a dovedností potřebných v průběhu praxe

7.1 Uplatněné znalosti a dovednosti

Již od samého počátku praxe jsem se potkával s pojmy a nákresy, se kterými jsem měl možnost se setkat již při studiu na VŠB-TU Ostrava zejména v předmětech Úvod do softwarového inženýrství, Vývoj informačních systémů a Úvod do databázových systémů. Získané vědomosti byly uplatněny zejména při analýzách, studiu dokumentace, podkladů a požadavků a nejvíce při návrhu, kde veškeré mnou vytvořené materiály byly, díky znalostem z vysoké školy, smysluplné i pro další členy týmu a prezentačně profesionální. Nejen při analýze a návrhu, ale i vlastní implementaci jsem zcela anebo částečně zúročil vědomosti i z dalších předmětů, konkrétně Algoritmy I a II, Programování I a II, Programovací jazyky I a II a Vývoj internetových aplikací. Především při zpracovávání úkolu č. 3 bylo obrovskou výhodou absolvování předmětů zabývajících se sítěmi, systémy a pomocnými utilitami, jmenovitě Počítačové sítě, Správa operačních systémů, Administrace databázových systémů a Skriptovací programovací jazyky a jejich aplikace. Během celé praxe jsem využíval i dovednosti získáné v Úvodu do teoretické informatiky, zvlášť části týkající se regulárních výrazů, a v menší míře také v Elektronickém publikování, to pak hlavně v závěru praxe. Nemohu nezmínit také Tvorbu aplikací pro mobilní zařízení I a II, kde znalosti a postupy osvojené v těchto předmětech byly velmi prospěšné jak při návrhu a implementaci webového API, tak i při vývoji mobilní aplikace.

7.2 Chybějící znalosti a dovednosti

V průběhu praxe jsem narazil i na záležitosti, které byly zmíněny jen okrajově anebo nebyly v osnovách povinných či povinně volitelných předmětů bakalářského studia vysoké školy vůbec. Programovací jazyk PHP není v rámci výuky bohužel zařazen ani okrajově, s MySQL jsem se na VŠB-TU Ostrava setkal pouze v jedné hodině ve volitelném předmětu Administrace databázových systémů. Ve velmi malé míře se na VŠB-TU Ostrava probíral verzovací systém Git, setkal jsem se ním pouze ve volitelném předmětu Tvorba aplikací pro mobilní zařízení II. Během celé praxe jsem se potkával s prostředím Linuxové konzole, bohužel i zde jsem mohl načerpat nějaké znalosti pouze ve volitelném předmětu Správa operačních systémů. S „kontejnery“, odlehčenou virtualizací a základní průpravou orchestračními nástroji, primárně Kubernetes a Docker, jsem se během mého pobytu na VŠB-TU Ostrava nesetkal vůbec. V mnou absolvované praxi jsem se však se všemi výše jmenovanými setkával denně.

8 Závěr

Absolvovanou odbornou praxi ve společnosti MonkeyData s.r.o. hodnotím velmi positivně. Zvolená praxe byla pestrá a svými úkoly hodně různorodá, i proto jsem se dříve pro tuto nabídku rozhodnul. Díky této zkušenosti jsem se podílel na vývoji skutečně prospěšných aplikací a rozšířil si tak obzory a upřesnil i doplnil znalosti získané během mého studia na VŠB-TU Ostrava. I přesto, že jsem se v průběhu praxe setkal s mnoha záležitostmi, se kterými jsem se na vysoké škole neseznámil, tak díky celkové připravenosti a výbavě ze studia jsem nebyl zaskočen a vždy jsem byl schopen se rychle zorientovat a nastudovat danou problematiku. Velice si také cením zkušenosti s prací v týmu, kde jsem si ve spolupráci se zkušenými vývojáři a kolegy programátory zlepšil své postupy při profesionálním vývoji software a nabyl i mnoha nových znalostí. Oceňuji mimo jiné i vřelé pracovní prostředí a skvělou firemní kulturu. Pevně věřím, že všechny tyto zkušenosti využiji jak při dalším studiu, tak také v budoucím zaměstnání.

Literatura

[1] *Build anything on android* [online]. [cit. 2019-04-01].
Dostupné z: https://developer.android.com/

[2] *Ansible - Automation for everyone* [online]. [cit. 2019-04-01].
Dostupné z: https://www.ansible.com/

[3] *Apache HTTP Server Project* [online]. [cit. 2019-04-01].
Dostupné z: https://httpd.apache.org/

[4] *Atlassian - Tools for teams, from startup to enterprise* [online]. [cit. 2019-04-01].
Dostupné z: https://www.atlassian.com/

[5] *Bcrypt* [online]. [cit. 2019-04-01].
Dostupné z: https://en.wikipedia.org/wiki/Bcrypt

[6] *CAPTCHA* [online]. [cit. 2019-04-01].
Dostupné z: https://en.wikipedia.org/wiki/CAPTCHA

[7] *Dependency Manager for PHP* [online]. [cit. 2019-04-01].
Dostupné z: https://getcomposer.org/

[8] *Docker: Enterprise Application Container Platform* [online]. [cit. 2019-04-01].
Dostupné z: https://www.docker.com/

[9] *Git - Distributed version control system* [online]. [cit. 2019-04-01].
Dostupné z: https://git-scm.com/

[10] *Google* [online]. [cit. 2019-04-01].
Dostupné z: https://www.google.com/

[11] *Java Software* [online]. [cit. 2019-04-01].
Dostupné z: https://www.oracle.com/java/

[12] *Jenkins - Build great things at any scale* [online]. [cit. 2019-04-01].
Dostupné z: https://jenkins.io/

[13] *Kubernetes: Production-Grade Container Orchestration* [online]. [cit. 2019-04-01].
Dostupné z: https://kubernetes.io/

[14] *Laravel - The PHP Framework For Web Artisans* [online]. [cit. 2019-04-01].
Dostupné z: https://laravel.com

[15] *MySQL - The world's most popular open source database* [online]. [cit. 2019-04-01].
Dostupné z: https://www.mysql.com

[16] *Nette* [online]. [cit. 2019-04-01].
Dostupné z: https://nette.org/en/

[17] *NGINX: High Performance Load Balancer, Web Server, & Reverse Proxy* [online]. [cit. 2019-04-01].
Dostupné z: https://www.nginx.com/

[18] *Percona – The Database Performance Experts* [online]. [cit. 2019-04-01].
Dostupné z: https://www.percona.com/

[19] *PHP: Hypertext Preprocessor* [online]. [cit. 2019-04-01].
Dostupné z: https://php.net

[20] *PHPUnit* [online]. [cit. 2019-04-01].
Dostupné z: https://phpunit.de

[21] *A type-safe HTTP client for Android and Java* [online]. [cit. 2019-04-01].
Dostupné z: https://square.github.io/retrofit/

[22] *RFC6749 - The OAuth 2.0 Authorization Framework* [online]. [cit. 2019-04-01].
Dostupné z: https://tools.ietf.org/html/rfc6749

[23] *RFC6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage* [online]. [cit. 2019-04-01].
Dostupné z: https://tools.ietf.org/html/rfc6750

[24] *Slack - Collaboration hub for work* [online]. [cit. 2019-04-01].
Dostupné z: https://slack.com/

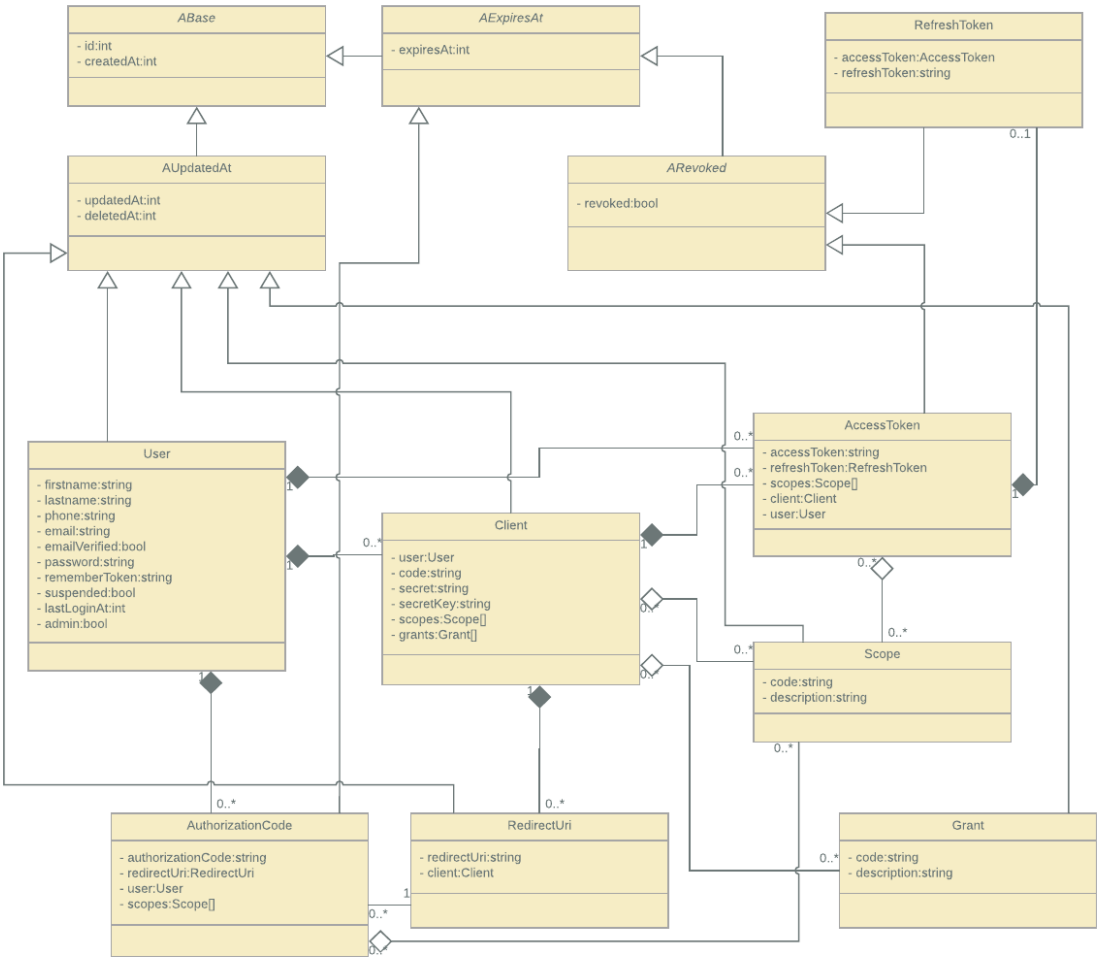
[25] *Symfony Framework* [online]. [cit. 2019-04-01].
Dostupné z: https://symfony.com/

[26] *Ubuntu* [online]. [cit. 2019-04-01].
Dostupné z: https://www.ubuntu.com/

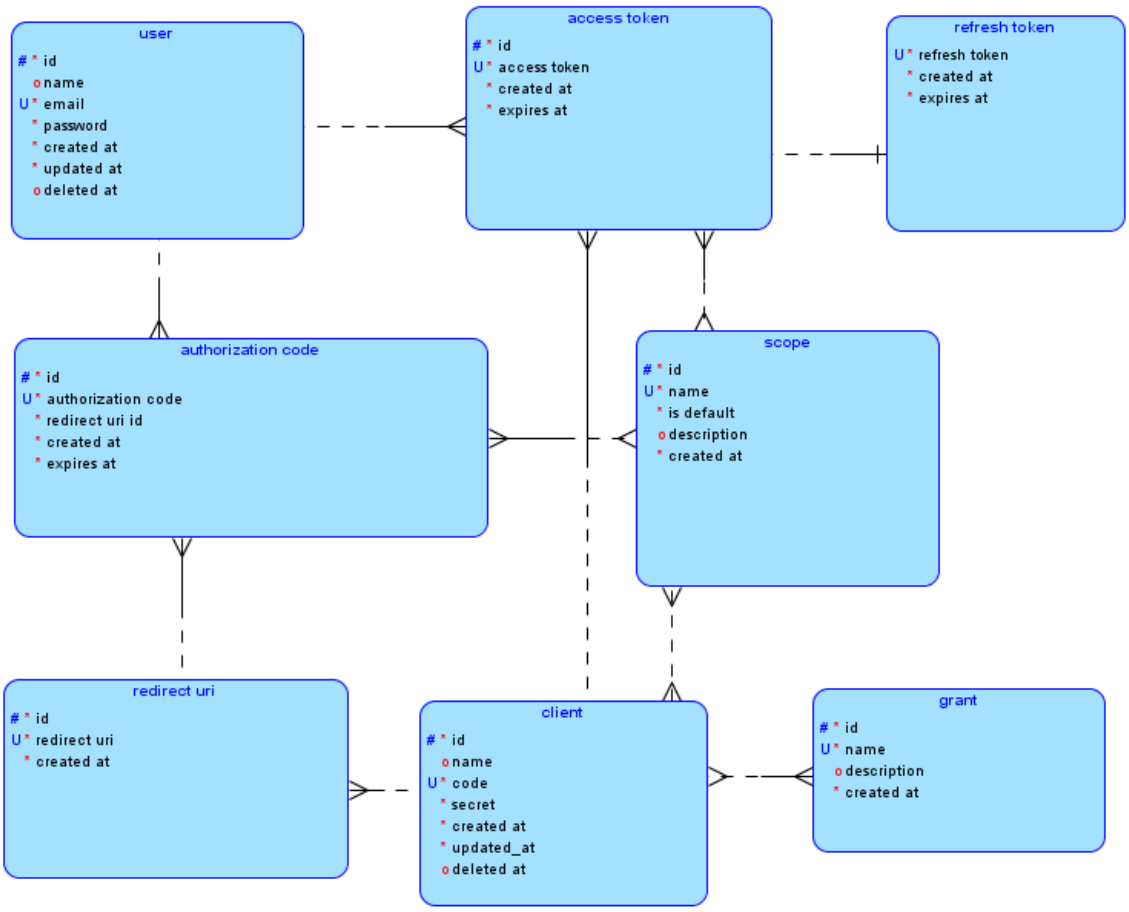
A Mezinárodní eCommerce akce

Název akce	Místo
eCommerce Shop Tech	Helsinky
eCommerce Berlin EXPO	Berlín
Webwinkel Vakdagen	Utrecht
Lightspeed Connect	Utrecht
Bigcommerce Palooza	Austin
DMEXCO	Cologne
eCommerce EXPO London	Londýn
IRCE	Chicago
Shopify UNITE	San Francisco
DLD Tel Aviv Innovation Festival	Tel Aviv

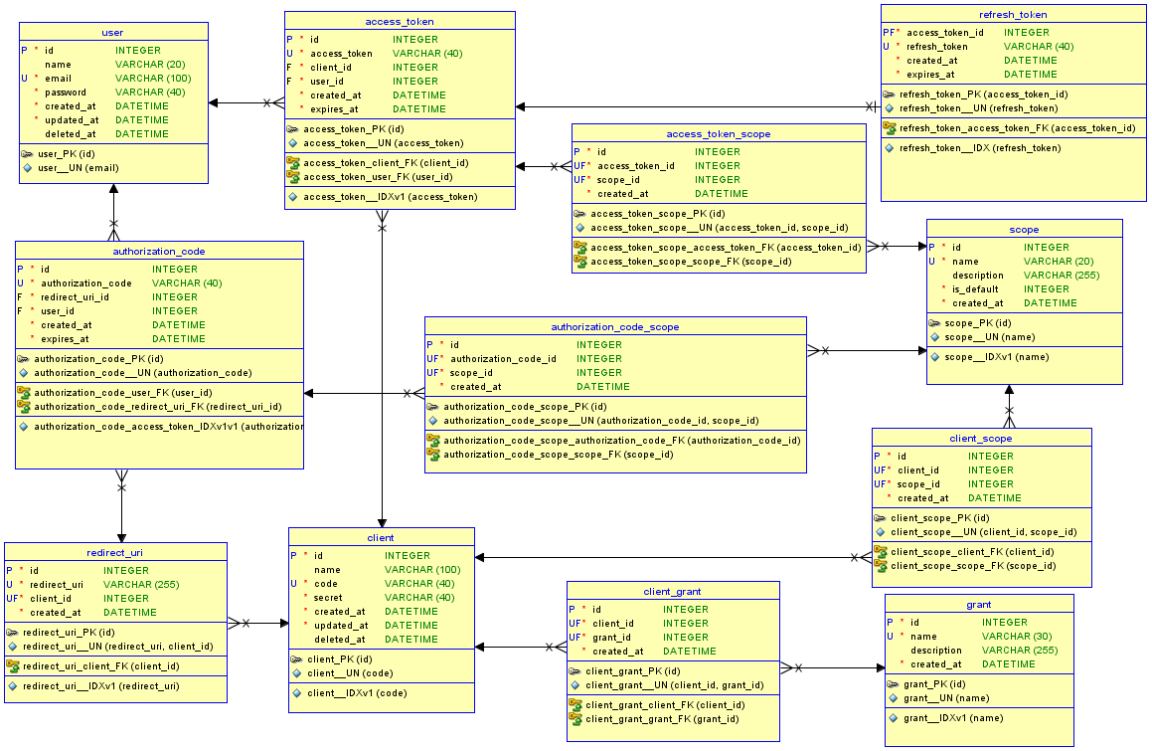
B Návrhové diagramy



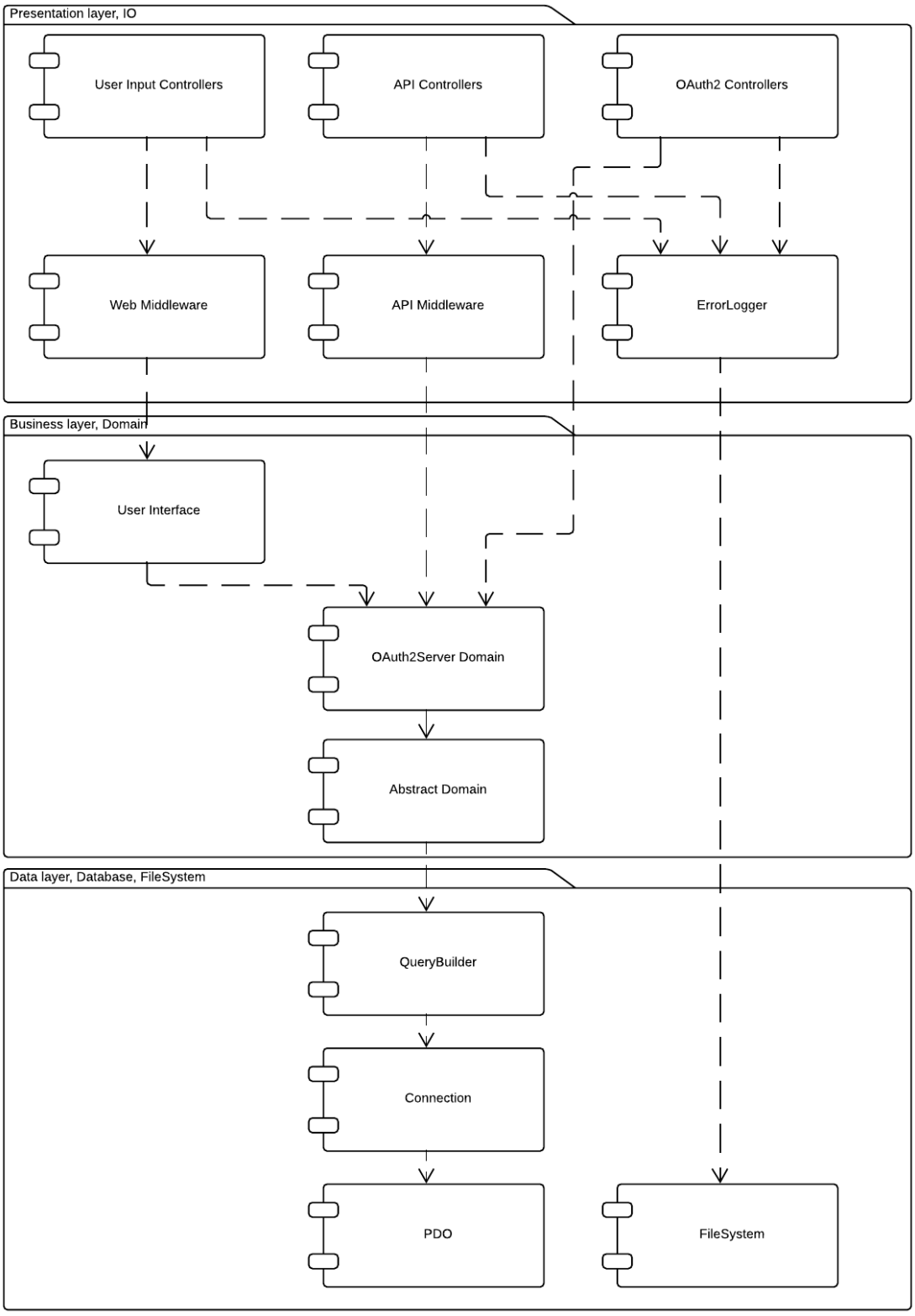
Obrázek 14: Diagram tříd



Obrázek 15: Logický model

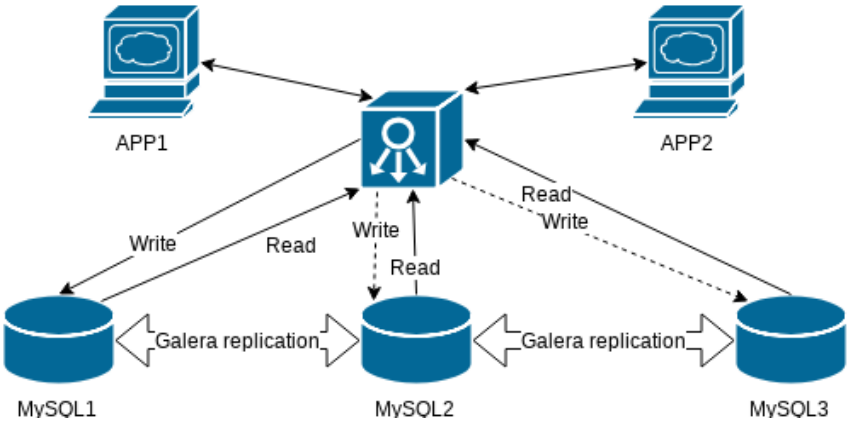


Obrázek 16: Relační model



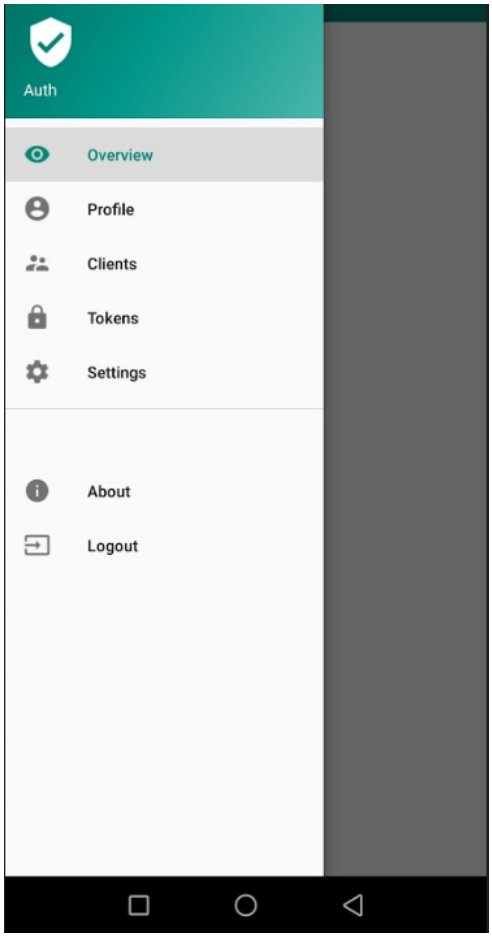
Obrázek 17: Diagram komponent

C Topologie úložiště

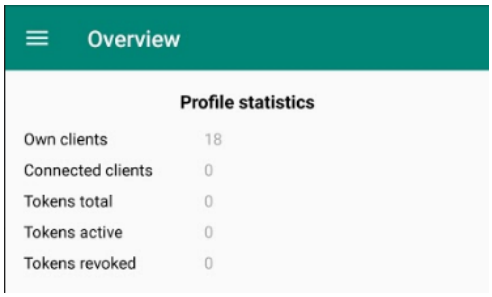


Obrázek 18: Galera replikace

D Mobilní aplikace



Obrázek 19: Boční ovládací panel



Obrázek 20: Základní přehled profilu

E Dockerizace

```
FROM mdUbuntu
RUN export LC_ALL=C.UTF-8 \
&& add-apt-repository -y ppa:ondrej/php \
&& add-apt-repository -y ppa:ondrej/pkg-gearman \
&& apt-get update
RUN apt-get -y --no-install-recommends install php7.1 libgearman7 libssl1.0.2
    php7.1-cli php7.1-common php7.1-json \
php7.1-opcache php7.1-readline php7.1-curl php7.1-mcrypt php7.1-xml php7.1-
    mysql \
php7.1-gd php7.1-intl php7.1-xsl php-oauth php-pear php7.1-bcmath php7.1-ldap \
php7.1-readline php7.1-tidy php7.1-xmlrpc php7.1-mbstring php7.1-soap php7.1-
    zip php-memcached \
php-imagick imagemagick php-gearman php7.1-cgi
RUN wget https://getcomposer.org/composer.phar \
&& chmod +x composer.phar \
&& cp composer.phar /usr/bin/composer \
&& rm composer.phar \
&& /usr/bin/composer self-update
RUN apt-get clean
```

Výpis 13: Předpis obrazu mdPHP

```
#!/usr/bin/env bash
set -e
phpVersion="7.1"
phpExec="php${phpVersion}"
dataSource="/data/source"
arguments="$@"
phpMemoryDeduction=${DOCKER_MD_PHP_MEMORY_GAP:-10}
if [[ -f "/.dockerenv" ]]
then
  DOCKER_MD="1"
fi
set +e
memoryLimit=$(cat /sys/fs/cgroup/memory/memory.limit_in_bytes 2> /dev/null)
set -e
if [[ -z "${memoryLimit}" ]]
then
```

```
  if [[ -z "$DOCKER_MD_PHP_MEMORY_LIMIT" ]]
  then
    echo -e "\e[0;31mPHP memory_limit NOT SET\e[0m"
    exit 1
  fi
else
  memoryLimitInMB=$((memoryLimit / 1024 / 1024))
  memoryLimitInMBForPHP=$((memoryLimitInMB - phpMemoryDeduction))
  DOCKER_MD_PHP_MEMORY_LIMIT="${memoryLimitInMBForPHP}M"
fi
echo -e "memory_limit = $DOCKER_MD_PHP_MEMORY_LIMIT" > /etc/php/${phpVersion}/
    cli/conf.d/memory-limit.ini
mkdir -p /etc/php/${phpVersion}/cgi/conf.d/
echo -e "memory_limit = $DOCKER_MD_PHP_MEMORY_LIMIT" > /etc/php/${phpVersion}/
    cgi/conf.d/memory-limit.ini
scriptDir=$(dirname $(readlink -f $0))
startAt=$(date +%s)
start=$(date)
echo -e "\e[0;33mStarted at $start\e[0m"
echo -e "Set to run at least \e[0;33m${minRuntime}\e[0m minute(s)\n"
function getAndPrintMaxMemoryUsageCgroup()
{
  set +e
  maxMemoryUsage=$(cat /sys/fs/cgroup/memory/memory.max_usage_in_bytes 2> /dev/
      null)
  set -e
  maxMemoryUsageInMB=$((maxMemoryUsage / 1024 / 1024))

  echo -e "\n\e[0;35mMax memory usage by cgroup: ${maxMemoryUsageInMB}MB (${
      maxMemoryUsage})\e[0m"
}
function exitHandler ()
{
  local errorCode="$?"

  finishedAt=$(date +%s)
  runningTimeInSeconds=$((finishedAt - startAt))

  echo -e "\n\e[0;33mFinished at $(date)\e[0m"
```

```
  echo -e "Running time: \e[0;33m${runningTimeInSeconds} seconds\e[0m"

  test ${errorCode} == 0 && exit 0

  getAndPrintMaxMemoryUsageCgroup

  echo -e "\n\e[0;31mFailure with error code ${errorCode}\e[0m\n"
  exit ${errorCode}
}
trap exitHandler EXIT
trap exit ERR
function runArtisan()
{
  ${phpExec} ${scriptDir}/artisan ${arguments}
}
function runServer()
{
  empty=""
  newArguments=${arguments/server/$empty}

  defaultConfig="--host=0.0.0.0 --port=8000"
  config=""

  ${phpExec} ${scriptDir}/artisan serve ${defaultConfig} ${config} ${
      newArguments}
}
if [[ "${arguments}" =~ "server" ]]
then
  runServer
else
  runArtisan
fi
getAndPrintMaxMemoryUsageCgroup
```

Výpis 14: Pomocný skript md-init